Crowdsourcing Cultural Appreciation for Flowering Trees in Costa Rica

An Interactive Qualifying Project Final Report Submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
In partial fulfillment of the requirements for the Degree of Bachelor of Science

Submitted by:

Jared Chan	**Robotics Engineering**
Joseph Dobbelaar	**Computer Science**
Alexis Graziano	**Biomedical Engineering**
Cole Parks	**Robotics Engineering & Computer Science**

Submitted to:
Professor James Chiarelli, Worcester Polytechnic Institute
Professor Bethel Eddy, Worcester Polytechnic Institute

Sponsors:
Diana Zuleta, Executive Director of Árboles Mágicos
San José, Costa Rica Project Center

WPI

ÁRBOLES MÁGICOS

Abstract

Árboles Mágicos is a non-governmental organization dedicated to spreading awareness and promoting the appreciation of flowering trees in Costa Rica. This project aims to develop a mobile application which involves crowdsourcing tree locations and photos to build a community around local flowering species and increase interest in them. Deliverables for this project were an Android mobile application, an administrator website, and a maintenance manual that enables future developers and Árboles Mágicos to understand how the app was designed and built as well as how they can further develop it.

Acknowledgements

We would like to express our gratitude to Diana Zuleta, the director of Árboles Mágicos, who worked closely with us to design the app, give feedback, and support us throughout the entire project. Diana, who is a professional designer as well, was a tremendous help in planning what went into the app and guiding us on how to effectively document our design choices for the benefit of future developers and the organization. Furthermore, she assisted us in contacting the former developer of the previous version of the app and provided us with a workspace that kept us productive. We were extremely pleased to have Diana as our sponsor's representative during the duration of the project.

We would also like to acknowledge the founder of Árboles Mágicos, Giancarlo Pucci, who has also provided detailed feedback about our work and has helped us in further understanding the motive behind the organization. Moreover, we would like to thank him for giving us the opportunity to work on this project. We also express our thanks to Matiaz, the developer of the previous app, who has helped us understand his previous work and how to continue it.

From Worcester Polytechnic Institute, we would like to thank our two advisors, Professors James Chiarelli and Beth Eddy. They have both supported us throughout our time in Costa Rica, having provided excellent feedback, and have helped a great deal in clarifying the direction of our project. In addition to our advisors, we heavily appreciate and express our thanks towards the Costa Rica IQP Project Center Director, Professor Melissa Belz, who made this project possible.

Lastly, we would like to express gratitude towards Worcester Polytechnic Institute for allowing us the opportunity to travel abroad to Costa Rica to work on this project. From this project, we have gained an unforgettable experience and newfound skills that will benefit us in the future.

Executive Summary

Árboles Mágicos is a non-profit organization dedicated to spreading appreciation of flowering trees throughout Costa Rica. Partnering with WPI, this organization hopes to upgrade their current mobile application to create an online community for users to connect.

Background

Costa Rica is known for being one of the most sustainable countries in the world. From 1940 to 1987, 72% of the country's forests were destroyed which left many problems including infertile land and a lack of clean water (Frost, 2020). To fix these issues throughout the country, Costa Rica's leaders set out to inspire change. Today Costa Rica houses about five percent of the planet's biodiversity and continues to preserve its land's natural beauty.

Árboles Mágicos is a nonprofit organization aimed at promoting cultural appreciation for the beauty of nature through the use of flowering trees. Through volunteers and sponsors, the organization has planted thousands of trees, educated students on the importance and beauty of trees, and released several products including multiple books to spread awareness and appreciation for flowering trees. Through their mission, Árboles Mágicos strives to encourage others to pause, connect, contemplate, and create.

Through the use of Árboles Mágicos' app, *Ojeadores* ("Scouts" in English), the organization developed a way for people to identify trees throughout Costa Rica. The first version of this app, available on iOS and Android, allows users to search for a species or use a color palette to match the flower color to a species. This made the app very user friendly and accessible to anyone despite their knowledge in trees. Users could also find a wide variety of information about each tree including photos, blooming months, sizes of the tree and flowers, and its origin. While this app was a useful tool for education, Árboles

Mágicos strived to create a community through the app, initiating the development of a second version. *Ojeadores* Version 2.0 implemented a map where users could add pins to showcase trees in their area. The pins consisted of photos uploaded by the user as well as the tree species and color. This was the social aspect of the application that Árboles Mágicos was aiming to create. However, due to the COVID-19 pandemic and a loss of funds, the project was halted and the application was left unfinished.

Methodology

Originally, the purpose of the project was to update the second version of *Ojeadores*, which went unreleased for iOS. However, due to our background research indicating a larger number of Android users in Costa Rica than iOS as well as a greater difficulty in developing for the latter, we proposed a change in the direction of the project to our sponsor: to instead remake the current version of *Ojeadores* for Android to increase the app's audience and produce a more flexible product that is easier to support in the future. Furthermore, in doing so, we still wanted to maintain the same ideals of the original app—to build a community around the contribution and celebration of flowering of trees in Costa Rica. To achieve these aims, we set the following objectives:

1. Gather Information and Requirements
2. Establish the Project Workflow
3. Design and Develop the Mobile Application
4. Create an Operations and Maintenance Manual

The first task in pursuing this project was to gather information and requirements for the app. Within the first two weeks of the project, we were given very little information on what features and work were done for the second version of *Ojeadores*. We received the resources and code for the second version of *Ojeadores* from the previous developer, yet were unsuccessful in getting the previous app to run properly. As a result, we decided to gather requirements and assess the work done on the previous app via informal discussion with Giancarlo Pucci, the founder, and Diana Zuleta, the director of Árboles Mágicos. Our goal for these discussions was to clarify how socially interactive the app should be, flaws of

the previous app, and desired changes as well as new features.

After we documented a list of requirements for the app, we laid the foundation for our project workflow. We decided to adapt two major frameworks for managing the development of an app as well as laying out the design: Agile Scrum and The Five Elements of User Experience (UX) Design. Agile Scrum is a framework for ensuring app development is organized, efficient, and iterative (James & Walter, 2021). As a team, we followed Agile Scrum by assigning features to work on at the beginning of each week, developing those features, and gaining feedback from Diana Zuleta at the end of the week. We iteratively performed the same process every week of assigning work and assessing feedback, developing, then asking for more feedback. The other framework we used, The Five Elements of UX Design pattern, breaks down app design into five layers: strategy, scope, structure, skeleton, and surface—which translate to gathering user needs and product goals, assessing how to meet those needs, making sure components of the app fit together, laying out where those components will go in the app, and fine tuning the small visual details of the app. Using this framework, we thoroughly planned and documented each layer of the app.

To develop and build the app, two pieces were essential: a frontend element—the visual, interactive aspect of the app—and a backend element—where and how the data for trees, users, and images is stored and retrieved. For the visual development of the app, we used Microsoft .NET MAUI, a software framework that makes supporting multiple platforms (iOS, Android, Windows, MacOS) easy and simple (Britch, Gechev, & Conrey, 2023). For the backend, we used Google Firebase, which provides online services for accessing and storing data (Firebase, n.d.-a).

In addition to the two aforementioned tools used, we also utilized another online service, GitHub, which stores code online, makes it accessible for others, and tracks the history of edits (GitHub, n.d.-a). In addition, we used GitHub Issues, which allows developers to create a bulletin board for laying out tasks to be done, assigning them, and collaborating on them (GitHub, n.d.-b). To write, run, and test code we used Microsoft Visual Studio, a platform for developing apps ((Microsoft, 2023). For making mockups for each page of the app, we used a simple and quick design tool called Figma (Figma, n.d.). Other tools we used were various APIs (Application Programming Interfaces), which helped provide services for

accessing Google Map data and assisted with checking if photo content was appropriate.

Once we had organized a list of user requirements (User Stories), accordingly decided on the functional components of the app to meet them, and knew which tools were needed, we started to design mockups using Figma. As a part of our end-of-the-week feedback from Diana Zuleta, mentioned earlier, we also received feedback on our mockup designs for each page of the app. Part of our iterative design process was to refine these mockups and re-present them each week until perfected. Once acceptable, we created the code that replicated the visual appearance of these mockups within the app.

After we finished our work on the app, including its pages and interactable features, the last step was to write an operations and maintenance manual that detailed our process, failures and successes, known issues, and the overall summary of how our code works. The following outline represents the main topics addressed in the manual:

1. Introduction to the project and our team (with contact information)
2. Software requirements and setup
3. App overview and structure
4. Common maintenance tasks
5. How to update the app
6. Finished, unfinished, and proposed features
7. Known issues
8. Long term associated costs
9. Publishing the app to the public

Findings

Due to our limited time frame of only eight weeks, our findings consist of feedback and the final Android application built iteratively over the project timeline. This feedback came from conversations with Árboles Mágicos about features they wanted, our own requirements gathering while designing the app's interface, and what we learned about software development through writing the app.

During the first two weeks, we spent our time speaking with Árboles Mágicos about what the app should do. We had concerns about the app becoming too focused on individual users trying to inundate the map with their own contributions. After several meetings, we elected to maintain the tree-oriented, community-driven vision that Árboles Mágicos held. We identified the important aspects of our user interface, created a list of requested features, and laid out how they would be implemented. We then spent time creating diagrams of how the user might interact with the app in order to create a positive user experience. By constructing a map of the entire application, we solidified our plan and began development.

We were initially unfamiliar with the tools we needed to write the app but, by the end of the term, we ported all required features to this Android application, along with several new features. A large portion of our findings stemmed from our experiences learning how to use the app development tools effectively. To document this process and ensure a smooth handoff to whoever takes over the project in the future, we delivered an operations and maintenance manual to Árboles Mágicos along with the code. This manual explains the structure of the code, the tools used (including any associated costs), known issues, suggestions for improvements, and tips for maintenance.

Conclusion

In conclusion, the development of the Android app and companion administrator website for Árboles Mágicos will contribute to the organization's mission of promoting public appreciation for flowering trees in Costa Rica. By allowing people to easily upload photos of trees to a map, the app provides a platform for individuals to contribute both in a scientific and a social sense. The iterative development process and forward-thinking design of the app and companion administrator website also ensure that they are not only currently functional, but also capable of future expansion and improvement. With these tools at their disposal, Árboles Mágicos can continue to make a significant impact in fostering a deeper appreciation for the natural beauty of Costa Rica's trees.

Authorship:

Note: Each chapter and section was reviewed and edited by every member of the team.

Abstract: Jared Chan
Acknowledgements: Jared Chan
Executive Summary: Jared Chan, Cole Parks, Lex Graziano, Joe Dobbelaar
1.0 Introduction: Cole Parks
2.0 Background/Literature Review
- **2.1:** Joe Dobbelaar
- **2.2:** Joe Dobbelaar
- **2.3:** Jared Chan, Lex Graziano
- **2.4:** Jared Chan, Cole Parks
- **2.5:** Cole Parks, Lex Graziano
- **2.6:** Jared Chan

3.0 Methodology
- **3.1:** Jared Chan
- **3.2:** Jared Chan, Joe Dobbelaar
- **3.3:** Jared Chan, Joe Dobbelaar, Cole Parks
- **3.4:** Joe Dobbelaar, Lex Graziano

4.0 Findings
- **4.1:** Jared
- **4.2:** Jared Chan, Lex Graziano, Cole Parks
- **4.3:** Lex, Joe
- **4.4:** Cole Parks, Lex Graziano
- **4.5:** Joe Dobbelaar, Jared Chan

5.0 Recommendations & Conclusion
- **5.1:** Lex Graziano, Cole Parks
- **5.2:** Jared Chan
- **5.3:** Lex Graziano, Cole Parks, Joe Dobblelaar
- **5.4:** Cole Parks, Jared Chan
- **5.5:** Cole Parks, Jared Chan, Lex Graziano

Appendix A: Developer Manual - Joe Dobbelaar

Table of Contents:

List of Figures:

Chapter 1: Introduction

Today, Costa Rica's economy stands on the back of its sprawling ecosystems. In addition to its many national parks, Costa Rica is home to about five percent of the planet's biodiversity. To protect the environment, laws have been passed to preserve these spaces—single-use plastics have been taxed, several geographical areas are protected, and the government gives tax incentives for sustainability. Additional laws have also been implemented to prevent and repair damages from mass deforestation (*Environment: Embajada de Costa Rica en dc*). These regulations have succeeded in making Costa Rica a world leader in sustainability. Clean air, national parks, lush rainforests, ecological tours, and environmentally conscious policies all play a part in making Costa Rica a well-loved tourist location.

One organization which is leading efforts to improve sustainability is Árboles Mágicos, a nonprofit organization devoted to expanding education about flowering trees in Costa Rica to bring about cultural change. Árboles Mágicos is involved with local reforestation efforts and offers professional development sessions to companies. The organization previously developed an app, *Ojeadores*, which helps users identify tree species by their flower color, and provides detailed information on the flowering season, origin, and location of various trees native to Costa Rica.

The aim of the next generation of Árboles Mágicos' app is to increase users' awareness and appreciation of flowering trees by providing a platform for social interaction and engagement around the various species that exist in Costa Rica. In doing so, the organization hopes to connect communities through the celebration of flowering trees and establish a mutually beneficial relationship between the two. The hope for the app is to grant individuals the ability to upload the locations of trees of interest, the names of the species, and photos of them. The main deliverable for this project is an application that crowdsources information from citizens to create a map of flowering tree locations—which motivates cultural change, conservation, and excitement about the environment of Costa Rica.

Chapter 2: Background

2.1 Motivation for Sustainability

Much of Costa Rica's rapid progress was inspired by collective trauma. Between 1940 and 1987, 72% of Costa Rica's forests were destroyed during an economic movement aimed at transforming forests into land that could be used for agriculture (Frost, 2020). Instead of benefiting from their newly cleared land, as the government had expected, Costa Rica was left with more problems than solutions. The new land quickly became infertile, clean water became harder to come by, and tourism fell. The lesson was succinct—Costa Rica's economy is propped up by the health of its ecosystems. Since adopting more eco-friendly policies, both Costa Rica's economy and environment have bloomed.

Since passing these ground-breaking environmental protection policies, Costa Rica has become a world leader in sustainability (Frost, 2020). This legislation has been welcomed by the population, in large part, due to their post-deforestation enthusiasm for preserving the land's natural beauty. This cultural shift towards sustainability defined the zeitgeist of the late 1900s and early 2000s, but with such rapid progress comes anxiety that the public will lose interest as the threat of deforestation fades from recent memory. Though Costa Rica has made clear progress towards combating climate change, the government recognizes that there is still more work to be done. With such sweeping changes and ambitious plans—like decarbonization by 2050 ("Costa Rica: The "Living Eden," 2019)—continued public support and cooperation is integral to maintaining forward progress.

2.2 Árboles Mágicos

Árboles Mágicos is an organization aimed at inspiring Costa Ricans to slow down and pause during their busy days to appreciate the beauty in nature that is sometimes overlooked. Their focus is on the appreciation of flowering trees. With a volunteer force of over three thousand members, the organization has planted more than thirty thousand trees, educated more than five thousand students, released several books, a decorative calendar, and a mobile app for identifying trees based on their flower color (Árboles Mágicos, n.d.). Árboles Mágicos works with other organizations to plan events and activities that teach people about the importance and beauty of trees. The focus of our project was to expand upon their mobile app to further spread their vision of how flowering trees can impact daily life.

Giancarlo Pucci, the founder of Árboles Mágicos, started the organization to inspire others and encourage them to find peace in the grasps of nature. Over a decade ago, Pucci was working in industry when he suddenly had the thought, "What am I doing with my life?" He had become unhappy with the way he was living and was striving for something more meaningful. Pucci was driving along the highway when he saw these beautiful flowering trees. This moment inspired him to pull over to the side of the road and experience the tranquility of the trees surrounding him. Pucci quit his job after and started this non-profit organization devoted to mapping and exploring the beauty of flowering trees in Costa Rica and soon all over the world (G. Pucci, personal communication, January 16, 2023).

The currently available Árboles Mágicos mobile app is called *Ojeadores*, translating to "Scouts" in English. Before starting our project, *Ojeadores* was strictly a tool for identifying trees. Users could select a region, browse tree categories, and identify whichever tree they wanted. While this app worked well as a tool for identifying trees, Árboles Mágicos wants to build a community around it. The plan is to create a mapping program where users can submit geo-location information for the flowering trees in their area, allowing anyone to be a contributor to the ever-growing tree identification community (G. Pucci, personal communication, January 16, 2023).

2.3 *Ojeadores*

2.3.1 *Ojeadores* Version 1.0

The first version of the *Ojeadores* app, which is available for both iOS and Android, identifies trees by having its users input the color that most closely resembles the shade of the leaves seen on the tree species in question. The app then presents a collection of all existing trees within Costa Rica that share a relatively similar shade of leaf color. Moreover, the app also informs users about the months during which a particular tree species blooms, how that species germinates, and the regions of Costa Rica in which it can be found (D. Zuleta, personal communication, December 7, 2022).

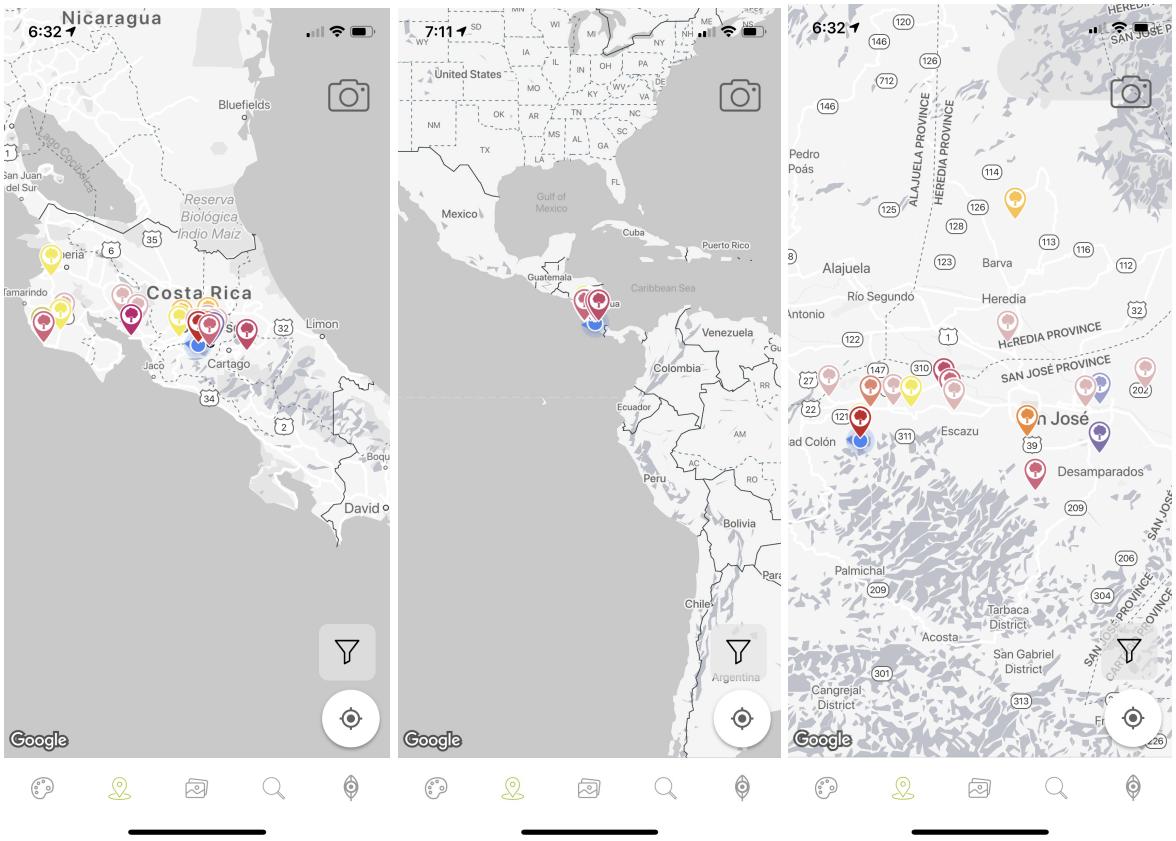
Version 1.0 of *Ojeadores* was only promoted at launch but has still managed to reach over 14,000 users (D. Zuleta, personal communication, December 7, 2022). Although *Ojeadores* achieves its goal of helping users identify flowering tree species, its method of identification does not guarantee highly accurate results. Furthermore, identification is entirely dependent on the user's ability to correctly recognize a flowering tree's leaf color. Thus, those who are color blind or have other vision impairments may not be able to reap the benefits of the current app design.

2.3.2 *Ojeadores* Version 2.0

Despite the initial success and outlook of the first version, Árboles Mágicos decided to re-evaluate the *Ojeadores* app because they felt the app was more of a brochure rather than an interactive tool to bring people together. In 2019, the organization contacted a developer to start programming a second version of the app for iOS only, which would add crowdsourcing technology and enable users to contribute to the mapping of flowering trees all across Costa Rica. However, in 2020, progress for the second version of the app was halted due to the Covid-19 pandemic and the app went unreleased (G. Pucci, personal communication, January 16, 2023).

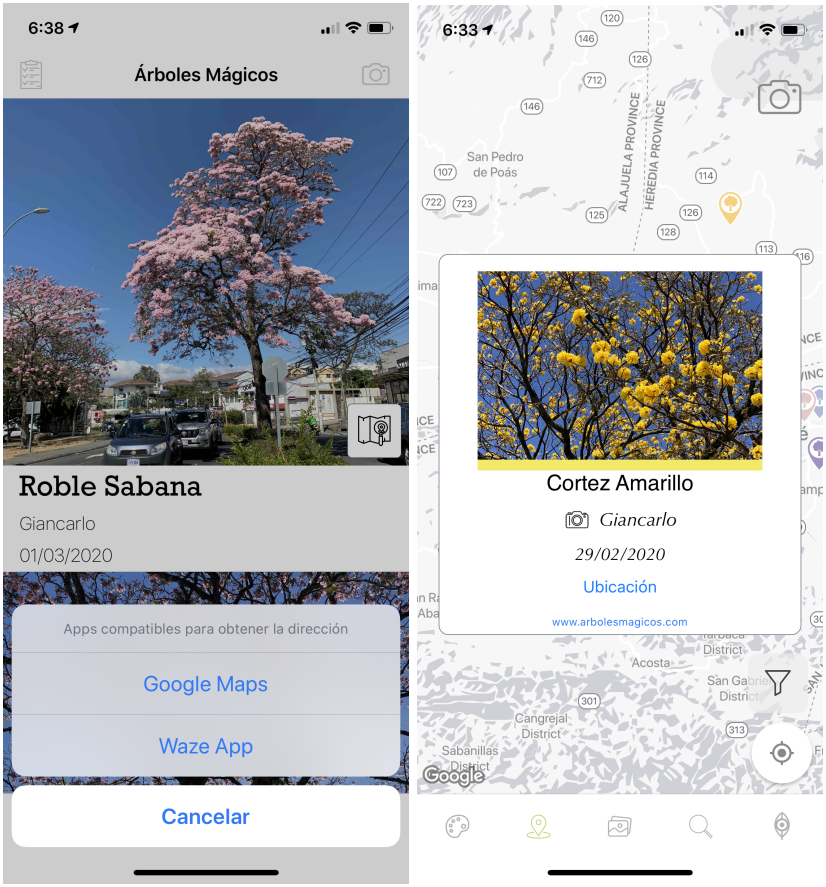
The goal of our project was to continue the development of the *Ojeadores* app based on where the previous developer stopped and prepare the app for a public release, preferably for Android as opposed to iOS. Moreover, the organization provided us with a list of requested features and design mockups to help us plan development. Below are images of the unreleased version of *Ojeadores*, from 2020.

Figure 1: *Ojeadores* Version 2.0 Map Interface



Note. The visual map interface for the second version of the *Ojeadores* app. Using the camera button, users can upload new information about a tree as well as its location. The map also has a filter button on the bottom right of the screen, which is above a button for centering the map view on the user's current location. From *Version 2.0* [Digital Image], by Árboles Mágicos, 2020. Reprinted with permission.

Figure 2: *Ojeadores* Pin Detail Modal



Note. The pin detail modal for the second version of the *Ojeadores* app. When users click on a pin, a small window appears that concisely displays information about that tree. The user can also expand that small window into a full screen view, which gives the option to use Google Maps or Waze for directions. From *Version 2.0* [Digital Image], by Árboles Mágicos, 2020. Reprinted with permission.

2.4 Other Mobile App Solutions

2.4.1 Plant Recognition Apps

LeafSnap, another similar plant identification app, uses image recognition via camera photos in conjunction with a plant database to assess the species of the plant in question. Moreover, the app relays information on how to plant and take care of different species (Appixi, 2022). In addition, vTree, an app produced by Virginia Tech, uses GPS features similar to Google Maps to locate and discover tree species nearby or far away. The app also factors in elevation and other key descriptive words that are entered by the user to help further narrow down the species yet to be identified (Peterson, 2020).

2.4.2 Crowdsourcing Apps

One feature which is very important and core to the next iteration of the *Ojeadores* app is the geolocation and crowdsourcing of flowering tree species data. This will allow users to view the locations of nearby flowering tree species on a map and hopefully encourage a stronger connection and appreciation for nature.

One example of crowdsourcing software is the SIMILE app, which uses citizen science to help monitor and report the condition of the water in lakes Maggiore, Como, and Lugano. To encourage participation in collecting and sharing data, the app employs several features that attempt to create a sense of community and involvement in lake life and preservation. For example, the app has a map, which lets the user see their current location as well as the location of their and other user's data entries.

Figure 3: SIMILE App Map User Interface

Note. A map screen mockup used to help visualize the original SIMILE app design. From "Crowdsourcing Water Quality With the SIMILE App," by D. Carrion, E. Pessina, C. A. Biraghi, G. Bratic, 2020, *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, *XLIII-B4-2020*, 245–25 (https://doi.org/10.5194/isprs-archives-XLIII-B4-2020-245-2020). CC-BY 4.0.

In addition, the app sends notifications to the user whenever others submit entries, organizes events, and creates announcements regarding the wellbeing of each lake. Users submit entries in the form of pictures they take with their phone's camera. To help dissociate the contents of the picture from the environmental conditions, such as rain, snow, fog, and bright or low illumination, the user can attach comments to their picture to help other users better interpret the results (Carrion et. al, 2020).

2.5 Citizen Science

Citizen science projects are effective tools for gathering data or performing various types of science since they employ the use of many people, accelerating the collection of data, which in turn can be used for research. In addition to the scientific benefits, citizen science can also benefit the citizens themselves.

For curriculum-based citizen science projects, the premise is generally experiential learning done by students who gather data to be used for the citizen science project. For example, Monarchs in the Classroom is a curriculum that teaches about monarch butterfly habitat, life cycle, and habitat loss while also gathering valuable data for scientists about eggs found on milkweed plants (Oberhauser & Caldwell, n.d.). Through this project, and similar projects, students are not just participating in scientific research, they are also contributing to a larger project, which in many cases relates to conservation, which in turn benefits entire species and habitats. While this positively affects the citizen science project, students are also motivated by the fact that the work being done has concrete effects on a real-life issue outside of their classroom.

The Cornell Lab of Ornithology has developed multiple apps for big data gathering of information related to birds, which has seen tremendous growth in recent years through the use of two apps: *eBird* and *Merlin*. *eBird* is a tool that birders can use to track sightings of birds, and includes a timestamp, location, number seen, and additional information that the birder can provide such as tag information and circumstance of the sighting. In 2021, *eBird* checklists reached over one billion records cumulatively from 723,000 *eBird* users (eBird, 2021). This abundance of information is used by the Cornell Lab of Ornithology to analyze trends that bird populations exhibit, aiding in conservation and law-making efforts in the US and other countries. The data is also used in their second app, *Merlin*, which is an app that identifies birds for users from a description of a bird they see, a photo taken of a bird, or a

recording of a bird. Not only does this in turn provide more data to the Cornell Lab of Ornithology, but it also teaches users about birds. In 2021, two million new users were added to *Merlin*, with an average of 700,000 active users each month, indicating not only that huge numbers of people are interested in furthering their understanding of birds, but that citizen science projects do lead to an increase in the citizen's understanding of science (eBird, 2021).

2.6 Mobile Platforms

The goal of the Árboles Mágicos' app is to bring people together to create a community surrounding flowering trees. Therefore, one of the most important decisions in developing the next version of the app is to choose a suitable mobile platform to build it on. Many developers typically choose to build their app for several different mobile operating systems, such as iOS and Android. However, because developing an app for different mobile platforms requires different resources, programming languages, and knowledge, attempting to make an app cross-compatible with varying platforms is time consuming and costly.

When it comes to the iOS platform, one major concern is how proprietary the software is. To create an iOS app, developers must use Apple's application development environment, XCode (*Xcode Overview,* n.d.). Moreover, to use XCode, the user has to own an Apple computer, and for the user to develop apps for the latest version of iOS, XCode must be on the latest release as well (Amazon, n.d.-a). Apple computers that are not up to date or too old in terms of model will not be compatible with the latest version of XCode, and therefore they cannot be used to develop apps for the most recent iOS version. As a result, development for iOS-based phones can be particularly challenging, especially when trying to develop and maintain apps that are expected to last for many years.

As for Android, it is an open-source mobile platform, which means there are many options in terms of development environments. Thus, access to Android development resources and tools is easier and more feasible. However, since the second version of the *Ojeadores* app has only been developed for iOS, focusing our resources towards providing an Android version of the app would call for recreating the current app from scratch.

For comparison and insight into the mobile phone market share in Costa Rica, data shows that around 74% of mobile phone users in Costa Rica have an Android, while 25% have an iOS device, as of January 2023 (Statcounter, n.d.). Overall, developing for the Android mobile market should yield a larger user base as well as easier access to resources. Therefore, having to rebuild the app for Android will be a better investment of our time.

Chapter 3: Methodology

The original purpose of this project was to increase the general public's knowledge, interest, and appreciation for flowering trees in Costa Rica through a new update to the *Ojeadores* app, which polished the current state of the app as well as introduced new features to make it more complete. However, because our background research indicated that more Android users exist in Costa Rica than iOS and that developing the app for the latter would present many challenges, we established a new direction for the project with our sponsor that would help them better meet their organization goals: to remake the second version of *Ojeadores* for Android and add new features if time allowed. By remaking the app, our goal was to provide Árboles Mágicos with a more flexible product that could easily be built upon in the future and one that would reach a wider audience. Yet, we still wanted to ensure the remake held onto the ideals of the original app—to build a community around the contribution and celebration of flowering of trees in Costa Rica. To achieve these aims, we set the following objectives:

1. Gather Information and Requirements
2. Establish the Project Workflow
3. Design and Develop the Mobile Application
4. Create an Operations and Maintenance Manual

3.1 Gathering Information and Requirements

3.1.1 Addressing Challenges and Concerns

One of the earliest challenges we encountered was the difficulty in accessing the previous developer's work, which prevented us from assessing the state of the *Ojeadores* and noting the necessary requirements to improve the newly rebuilt app. As discussed in the background chapter, iOS is particularly challenging in nature to work with due to the need for an Apple computer that is up to date and has the latest version of the programming software, XCode. Although one member in our team had an

Apple computer with the aforementioned specifications, we still could not set up the previous code to run the most recently updated version of *Ojeadores*, partly due to the previous developer leaving almost no documentation about how to use, set up, and run the code. As a result, we had to use images taken of the previous app, instead, which were few. With the lack of knowledge about the previous app's state, we noted the few features and visuals present within the pictures given to us and decided to base most of the requirements gathering on informal discussions with the organization's director, Diana Zuleta, and its founder, Giancarlo Pucci.

3.1.2 Conducting Informal Discussions with Árboles Mágicos

Since we had no written documentation with justification for the design choices made for the previous app, our main goal of these informal discussions with Diana and Giancarlo were to determine how much social interaction should occur within the app, flaws of the previous app, and features that should be made differently or were missing before.

We knew beforehand that the app's purpose was to build a community around flowering trees; however, it was ambiguous whether the app should behave like mainstream social media apps, with the ability to follow others and react to posts or should reduce user interaction to only be necessary when users make scientific corrections or comments on posts.

We also wanted to note a rough count of how many features they were requesting and how many were considered absolutely necessary to include in the end product at the end of the seven weeks we had to work on the project. Each week that we met with Diana Zuleta to gain feedback, we constantly re-evaluated how many features we could accomplish in our remaining time in Costa Rica. Even if we thought we did not have time to add certain features discussed, we made sure to document the importance of those features and list ideas for implementing them in the app. Because of the documentation we laid out on features that did not come to fruition, future developers and Árboles Mágicos will still be able to return to them in the future for re-evaluation.

3.2 Establishing the Project Workflow

Building apps, especially at the industry and commercial level, is a complex process which requires careful thought and planning. Thus, systematic organization and methodology are necessary to ensure that we not only produce a successfully interactive, social, and scalable app, but we provide structural documentation that enables Árboles Mágicos and future developers of the app to understand our design decisions and development process (D. Zuleta, personal communication, January 27, 2023).

3.2.1 Project Management Frameworks

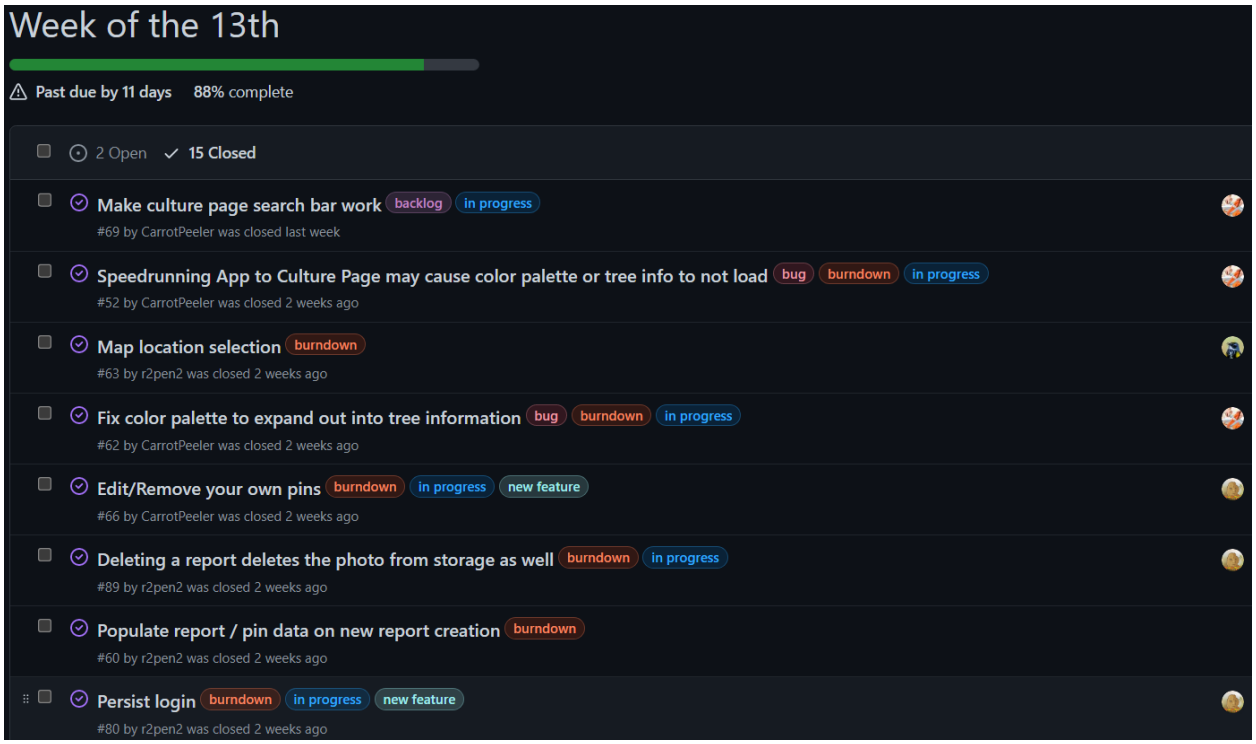
With some of our team members having taken courses in application design beforehand, we agreed that following a structural framework for application development would help us stay organized, focused, and productive throughout the course of the project. Since the amount of time we had to develop a port of the *Ojeadores* app for Android was limited, we decided to follow a common project management framework that three of the four of us have worked with in the past, Agile Scrum.

Agile Scrum organizes workflow for an entire project by establishing set periods of time where development occurs, called Sprints—which are essentially deadlines. When planning sprints, the team first develops a list of product requirements in the form of User Stories. A User Story, typically written on an index card, represents a single action or feature the user requires the app to have. These requirements are written from the perspective of the user and include user justification as well to help developers understand how to create functionality that meets the needs of the user (James & Walter, 2021). For example, a User Story might state "As a user, I want the ability to flag a photo, so I can report inappropriate pictures and content." Adding justification to User Stories is crucial, especially in our case, so Árboles Mágicos and future developers understand our design decisions. Within the first two weeks, our main goal was to analyze the list of requirements and needs mentioned in discussions with Diana and

Giancarlo and derive User Stories from them. In later sections, examples of User Stories we created are shown and our thought process is explained in further detail.

After a list of User Stories is written out and organized, the next step is to assess the priority and difficulty for each, and to group User Stories into Sprints based on these qualities (James & Walter, 2021). To follow this idea, at the beginning of each week, we looked at our written list of User Stories, determined which ones we were going to accomplish for the week, and divided them up among ourselves. While planning the User Stories for the week, we took into consideration several factors: how essential the feature was to the app, how long and difficult the feature was, and if the feature required lots of feedback and heavy consideration. For example, we believed the map along with the buttons and features within it were core features of the app and required lots of feedback from Diana. Thus, we grouped User Stories related to map features together and spent the first few weekly Sprints working on those particular elements. As an example, the figure below demonstrates the Sprint planning we did for the week of February 13th.

Figure 4: Sprint Log of User Stories to Accomplish



Note. The above Sprint planner for the week of February 13th was made using GitHub Issues. The bar at the top indicates the progress of how many User Stories have been completed and sections below are individual User Stories. When a User Story is completed, it is marked as "closed." Conversely, "open" indicates User Stories are unfinished. Usage of GitHub Issues and the software is elaborated on further in section 3.3.2. Own work.

Normally Sprints range anywhere from a single day to an entire week, but our sprints lasted a week and sometimes even longer due to setbacks and troubleshooting. However, regardless of delays, sprints kept us on track to finish features every week and helped us stay alert of our overall progress on the app.

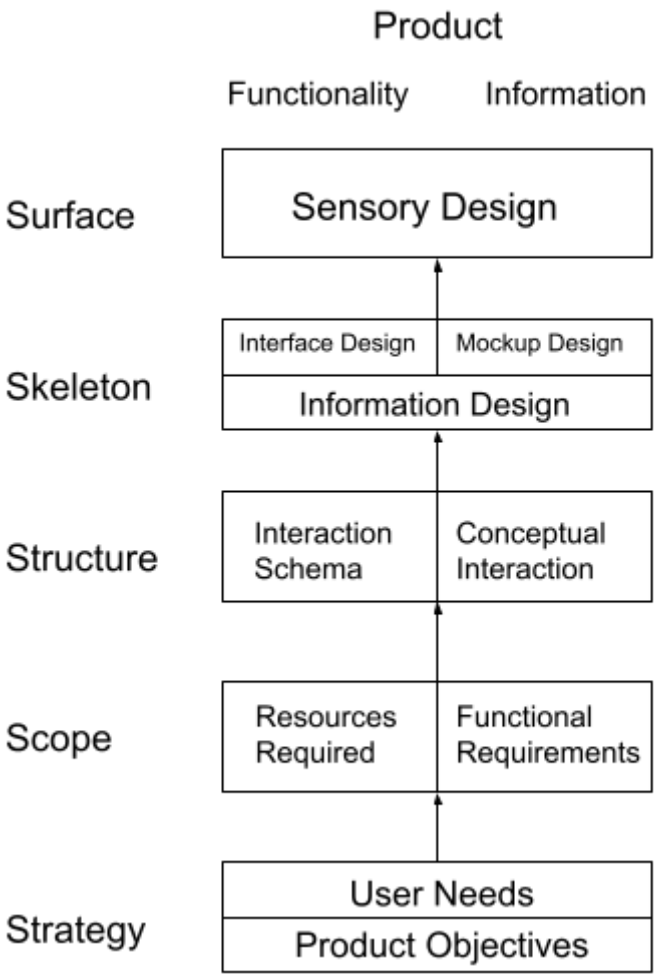
Agile Scrum promotes an iterative development process, where the work produced each week is built upon the previous weeks' work and is refined through feedback and review (James & Walter, 2021). Aiming to have a somewhat finished app at the end of each week allowed us to demonstrate the functionality of the features we were implementing in a very rough form to our sponsor's representative,

Diana. In compliance with Agile Scrum, we incorporated the feedback collected at the end of each week as well as other User Stories into the next weekly Sprint. Because of the cyclical process of planning weekly Sprints, accomplishing User Stories, and collecting feedback at the end of each week, Agile Scrum enabled us to incrementally polish and sculpt the overall design of the app over the seven weeks, rather than in the last week.

3.2.2 Structural Design Frameworks

While Agile Scrum is a useful tool for planning weekly project goals, collecting feedback, and guiding iterative development to ensure smooth and successful progress, we needed a framework for specifically guiding the design process of the app. Based on a strong recommendation from our sponsor, Diana Zuleta, we adopted the Five Elements of User Experience (UX) Design methodology, which laid out a structured order of steps for designing the user interface for the Android application. The figure below conveys the building blocks and order of steps involved in the Five Elements of UX Design methodology.

Figure 5: The Five Elements of UX Design Diagram



Note. Our interpretation and simplification of the Five Elements of UX Design diagram, inspired from the book, *The Elements of User Experience*, by Garrett, J. J. Own work.

The goal of the Five Elements of UX Design methodology is to start with abstract design (e.g., brainstorming and rough sketches) and gradually move towards concrete implementation, which is the actual interactive app. To follow this concept, we started at the first layer, strategy, by drafting a list of potential user needs as well as product objectives in the form of User Stories, as previously discussed.

From establishing a list of User Stories, we were able to assess the features necessary to meet the needs described by the User Stories, which tackled the scope layer. For example, if a User Story stated that the user wanted to report inappropriate content, we needed to implement a button that the user could press to report a photo as well as an entire moderation system that would enable admins to see reported content and handle it appropriately.

To determine the next aspect of design structure, we conceptualized how all the assessed features would interact with each other and potential issues that may arise as a result. Since trying to address how all the components of an app interact with each other is complex and would not be easy to understand for others besides ourselves, Diana Zuleta requested that we design a flow diagram with arrows to showcase visually and clearly how a user would navigate through the pages of the app. Making the flow diagram involved creating mockup designs of each page in the app using a software mockup tool called Figma, further discussed in section 3.3.2. A representation of the flow diagram is shown and elaborated on in section 4.1.2.

The last steps involved in The Five Elements of UX Design were planning the skeleton and surface layers of the app, which dealt with the layout of visual elements on each page of the app as well as the color scheme, selection of word choice, and final touches on the overall appearance and feeling given off by the design of the interface. Both of these layers were initially addressed during the creation of the flow diagram and were fully fleshed out during the final week of the project, after receiving extensive feedback from both Diana Zuleta and Giancarlo Pucci on how specific pages should appear.

3.3 Designing and Developing the Mobile Application

3.3.1 Mobile App Architecture

Since the app involves crowdsourcing flowering tree data, which includes photos, locations, species, color, etc., we needed to create an app which could accomplish two large tasks:

1) Store and retrieve user/organization data (backend)

2) Display user/organization data (frontend)

With background knowledge from previous courses taken at WPI, we knew we needed to create a backend and frontend component to the app to accomplish the two aforementioned tasks. The backend component is the database, where all the application data is stored and retrieved. Conversely, the frontend is what users see when they use the app, also known as the user interface: buttons, pages, photos, a map, etc. Therefore, since the frontend and backend are two separate entities that interact with each other, we need to use two different pieces of software, accordingly.

For the frontend, we chose to use Microsoft's software framework for developing user interfaces, .NET MAUI. We specifically decided on MAUI because it is a relatively new user interface framework, which enables an app to be developed for multiple platforms (iOS, Android, Windows, MacOS) under one codebase (Britch, Gechev, & Conrey, 2023). Based on common knowledge in programming, when an app supports multiple platforms, the developers normally need to program the entire app from the ground up for each platform. However, the cross-platform nature of MAUI eliminates the tedious and redundant work involved in doing so because the developers only need to write one set of code for the app. By using MAUI, we initially built and tested the app for Android but left the possibility for future developers to expand the app to many other platforms, such as iOS, with ease.

We used Google Firebase to build the backend, which was also used by the previous developer. Firebase allowed us to quickly implement secure user authentication, a database to store our records, and a place to store uploaded photos (Firebase, n.d.-a). Firebase also allowed us to set up live updates so that, when a database object changed for any reason, everyone with the app open would receive that change automatically. The Firebase free tier, called the “Spark” plan, was more than enough for our purposes during development. As the app scales up, however, Árboles Mágicos may have to invest in a paid plan. Higher tier Firebase services and their associated costs are explored further in section 4.5 (Scalability).

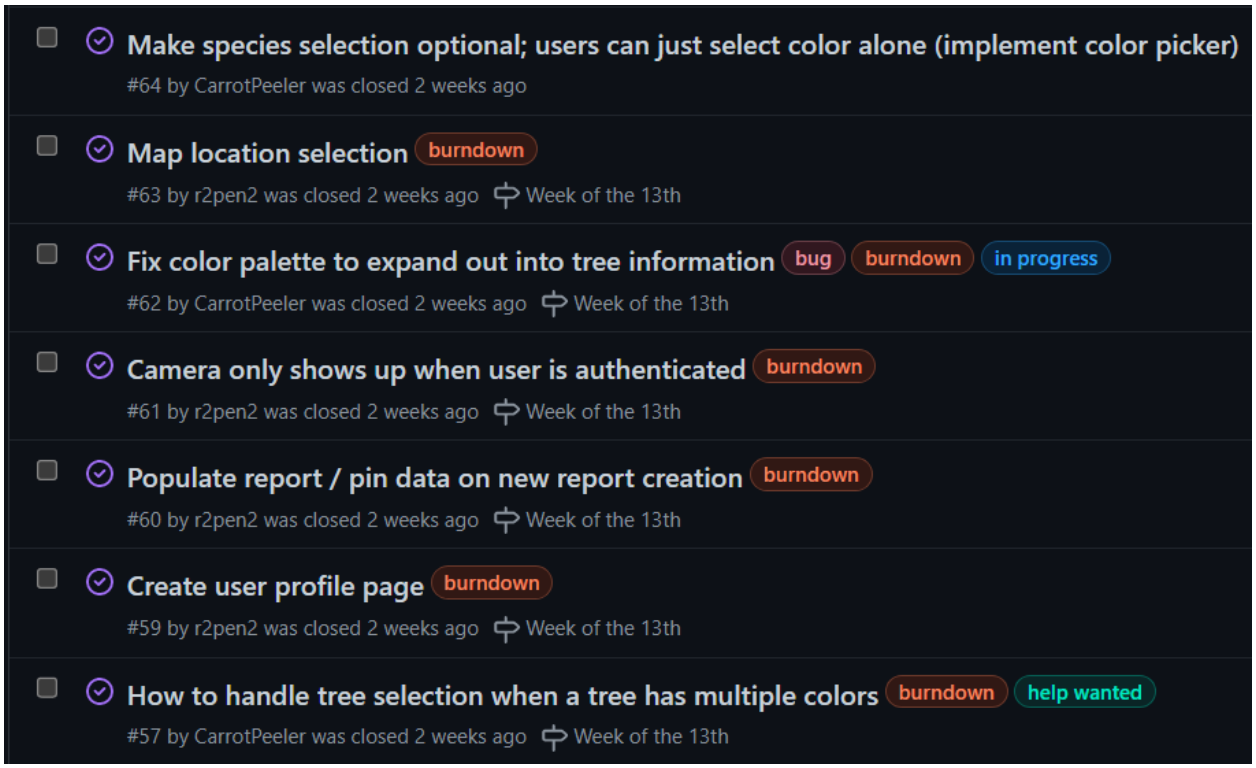
3.3.2 Development Tools

To further assist our development on the frontend and backend of the app, we used several other software tools, some of which were platforms for development and others being services called Application Programming Interfaces (APIs), which let us use pre-programmed functionality in our own project.

For storing and tracking the version history of our written code online, we used GitHub, which is a widely-known source control platform that makes it easy to document, share, and contribute to codebases (GitHub, n.d.-a). The platform is comparable to Google Drive in terms of how it is utilized collaboratively, allowing users to mark up code with labels, comments, and suggested edits. We also used GitHub to deliver the final codebase in addition to the maintenance manual to Árboles Mágicos.

Among many of the features built into GitHub is a ticketing system, GitHub Issues, which grants developers the ability to create tickets. Tickets are a powerful means of communicating with other developers working on the same project. They contain a small description of an issue with the code, a new feature to develop, or a general announcement for the project (GitHub, n.d.-b). By using tickets to create a bulletin board of tasks for the team, we were able to track our progress, which was shown in Figure 4. In addition, the figure below shows another view of the GitHub Issues bulletin board.

Figure 6: GitHub Issues Bulletin Board Feature



Note. As shown, developers can attach labels to each ticket to make them more distinguishable. Labels displayed above, such as *burndown* indicate that the ticket needs to be completed by the end of the week. *Bug* indicates an issue found; *in progress* means a ticket is currently being worked on; *help wanted* alerts that multiple developers need to address a ticket. Own work.

Using GitHub Issues, we organized a roadmap for all the work yet to be done and were able to effectively categorize different types of tasks and make deadlines and priorities discernable. From the figure above, we used labels, such as “burndown,” to add urgency to features we needed to complete by the end of the week of February 13th. Furthermore, by labeling tickets, future developers and the organization can see ongoing problems with the app and understand which features are unfinished (GitHub, n.d.-b).

To write, build, and test the app, we used a commonly popular code development platform, Microsoft Visual Studio. Much like Microsoft Word, Visual Studio contains powerful tools for designing

graphics, testing, and writing code, such as a code correction feature that acts analogously to grammar correction for Word (Microsoft, 2023). Visual Studio helped us organize documents and perform quick edits using its AI correction tools. Moreover, the platform was the foundation on which the entire project was built because it housed all the project resources and had built-in integration with almost all the other tools used for the project.

When it came to designing mockups, we used an online tool, Figma, which enabled us to quickly overlay shapes and patterns to represent the different pages of the app and create a flow diagram for all of them (Figma, n.d.).

Lastly, to reduce our workload, we made use of pre-existing service programs known as Application Programming Interfaces (APIs). An API is essentially a messenger, which other programs use as an intermediary for communication (Amazon, n.d.-b). For this project, we made use of two APIs, the first being Google's API library, which permitted us to use Google's map data and resources as well their Places service for looking up addresses and converting them to world coordinates. Additionally, we used the Cloudmersive API for determining whether user-uploaded photos contained sensitive or inappropriate content (Cloudmersive, n. d.-a).

3.3.3 User Interface Design

The user interface is one of the most important aspects of an app—it embodies the visual appearance of the app and is the primary way through which users interact with it. If the users do not like how the app looks, or find navigating through the app to be difficult or to feel like a chore, the user interface fails to do its job, which results in users no longer wanting to use the app. Since the user interface is the main attraction of the app, we put a lot of careful thought into how users navigate through the app, as well as how different features are accessed within the app. Our main priority was to make the features within the app as easily accessible as possible in addition to being intuitive to the user. To quickly visualize the pages that we were going to develop in code and to ensure the navigation flow between

pages made sense, we used Figma to design user interface mockups of the pages before implementing them in code (Figma, n.d.). By following the aforementioned Five Elements of UX Design pattern, we had already gathered all the required features for the app and assessed the corresponding functional components necessary to accommodate them. Thus, because we analyzed and documented a list of functional components each page required, we were able to clearly understand what visual components were essential for every page within the app, making designing mockups quick and efficient.

The next step was to gather feedback on these designs from our sponsor, Árboles Mágicos. We met with Diana Zuleta on a weekly basis, where we demonstrated new mockups that incorporated feedback from the previous weeks. Positive and negative critiques about the visuals and functionality of each page, as well as future possibilities were recorded in our meeting documents. After gaining weekly feedback, we implemented our user interface model using the Microsoft .NET MAUI framework. This process was iterative and happened every week, to incorporate as much feedback as possible. As a result, after seven weeks, we ended up with heavily refined designs for each page within the app that were appealing and intuitive to the users.

3.4 Creating an Operations and Maintenance Manual

In order for the app to be maintainable and updatable, we created an operations and maintenance manual (O&M), which lives within the online GitHub repository where the codebase is stored. This is a document that gives detailed instructions and guidelines for developing and updating the app over time. Maintenance manuals are crucial for development because it is very difficult to continue working on an undocumented project with many parts. The manual outlines a standard for working on the app, which when followed, helps to maintain consistency in the code design. In addition, this facilitates training new developers on the standards of development used for this project, ensuring that even through developer changes, the code structure and maintenance stay relatively clean and consistent.

The maintenance manual is organized with this general structure:

1. Introduction to the project and our team (with contact information)
2. Software requirements and setup
3. App overview and structure
4. Common maintenance tasks
5. How to update the app
6. Finished, unfinished, and proposed features
7. Known issues
8. Long term associated costs
9. Publishing the app to the public

The aim of the Operations and Maintenance manual is to ensure that the application can be easily updated, maintained, and added to in the future. We made the manual as clear and concise as possible so that future developers and Árboles Mágicos can understand how the app was assembled, why we approached the development process the way we did, and why we opted to use certain resources and tools over others.

Chapter 4: Findings and Analysis

The two main deliverables that Árboles Mágicos received upon the completion of the project were the finished Android mobile application and the operations and maintenance manual. By following the Agile Scrum methodology, which helped us plan the project flow and manage our time well, and designing around the Five Elements of User Experience, which guided the design process, the structure and appearance of the app changed drastically throughout the seven week period of development. Each week, we carried out an iterative development process, working closely with our sponsor to plan the flow and logistics of the app as well as our weekly goals. Our results are heavily qualitative since our work involved in-depth discussions and feedback about app structure, design, moderation, crowdsourcing, and building a community, which were the fundamental goals of this project.

4.1 Results from Informal Discussions with Árboles Mágicos

In the first two weeks of work, we discussed with Diana Zuleta and Giancarlo Pucci the core aspects of the app as well as desired features that should appear in the newly rebuilt version. During the very first discussion we held with them, we learned that in addition to the onset of the COVID-19 pandemic, a major reason why the second version of *Ojeadores* went unreleased was due to a severe lack of moderation features. Giancarlo Pucci emphasized having different levels of permissions within the app, such as admin and moderator roles, as well as methods for reporting inappropriate photo uploads and flagging incorrectly identified labeled species or colors. To address these concerns, we proposed an administrator portal for handling and verifying flagged uploads in addition to banning users from posting.

Giancarlo Pucci also touched upon how he envisioned the app to be a social network, where people could react and comment on other users' uploaded photos (G. Pucci, personal communication,

January 16, 2023). In response to creating a social network around the app, we developed a mockup prototype of a “Feed” page, where users see the latest posts from others. However, after reviewing the idea with Diana Zuleta in the following week, she corroborated with Giancarlo that the app should not feel similar to a mainstream social media app. Instead, it should emphasize the viewing and exploring of flowering tree culture with support from others in the community. In addition, Diana Zuleta suggested that the feed page become a bulletin board with recently-posted events and activities by the organization and highlight certain posted trees every week. Furthermore, she stated that a social-media-esque application would encourage users to focus on the photo-taking aspect of the app, which could lead to a competition for likes and reactions. (D. Zuleta, personal communication, February 2, 2023).

In addition to the feed page, the existing color palette page needed to be addressed. Below is a figure representing the color palette page from the second, unreleased version of *Ojeadores*.

Figure 7: *Ojeadores* 2.0 Color Palette Page

Note. Because of the appealing visuals, the color palette page invites users to explore the different colors of flowering trees. When tapping on a color, the associated flowering trees that share that color appear as a list and the user can tap on an individual species within the list to further learn about it. From *Version 2.0* [Digital Image], by Árboles Mágicos, 2020. Reprinted with permission.

We collectively decided that the design of the palette was an inconvenient way to explore tree species for two reasons. First, from a technical perspective, scrolling through the palette to find a particular color was tedious. Multiple color headers could be expanded at once, cluttering the screen and requiring users to re-select those specific color headers again to close their respective lists of trees. Second, both versions of *Ojeadores* contained a second page in addition to the color palette–the search page–which let users search for tree species by common name. Our initial design choice involved combining the two pages into a single page, enabling users to search by name and discover by selecting a color in one page. When we mentioned these considerations to Zuleta, she firmly believed that the palette

should maintain the same design because of how user friendly, simple, and inviting it is (D. Zuleta, personal communication, January 26, 2023). As a result, we kept the same color palette design but with the addition of a search bar, eliminating the need for a separate species search page.

After our last preliminary meeting with Diana Zuleta and Giancarlo for clarifying project requirements, we concluded that the bare minimum features to include in the final product of the app were pages for the color palette and species discovery, the map, and events and announcements. Additionally, derived from the requirements of having users upload photos and interact with the organization via events, we decided that there should be an account page for users to view and edit their personal data and a page about Árboles Mágicos, which contains links to their social media, website, and contact information.

4.2 Results from the Design Process

When we first began planning the design of the app, we directed most of our focus towards the finer details of the app, such as the visual appearance of each page on the app and the functionality of the map. However, by talking with our sponsor, we came to understand that the foundation of development and design was not solely based on mockups and diagrams, but involved abstract concepts as well, including discussion about what the users may want from the app and how we can align their needs with the goals of Árboles Mágicos. Utilizing the Five Elements of User Experience, we drafted documentation that reflects the process we took: starting from abstract notions which build into concrete products, such as a working Android version of the crowdsourcing app and the maintenance manual.

4.2.1 User Stories

Part of the design process is having a clear list of features and reasonings. We developed a list of User Stories that highlighted small but important features as well as major concepts that users would like

to have in the app. User Stories, which are written by a developer but use the perspective of a user, are typically used in app creation to help the developer better understand how a user with no technical background would want to use their application. While we developed User Stories for each page of the app, the User Stories for the map page are listed below as an example.

As a user, I want to be able to add a tree pin to the map, so I can contribute and feel part of a community.

As a user, I want to be able to remove a pin I made, if I made an error or want to remove my entry.

As a user, I want to be able to view other people's pins w/ tree details, etc., so I can view the other entries in the community.

As a user, I want to filter pins by flowering tree color b/c this is the simplest category to sort trees by.

As a user, I want to filter pins by tree species, so I can understand the different types of trees in the area.

As a user, I want to filter pins by user; certain users may upload higher quality pictures that interest me or contribute largely to the map.

As a user, I want to filter pins by location, so I can see trees in my area or in places I will visit.

As a user, I want to be able to clearly see pins, not overlapped clusters, so I am not overwhelmed.

As a user, I want to be able to get directions to a pin, so I can walk/visit that tree.

As a user, I want to be able to navigate an optimal route/path to multiple pins, so I plan a trip to see multiple trees efficiently.

As a user, I want to be able to find my location on the map at any time, so I can see where I am in relation to other pins.

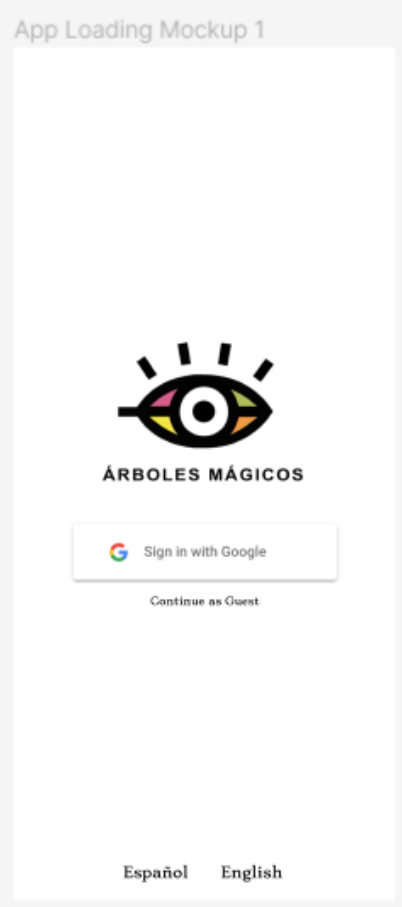
The user stories above highlight needs of the user. For example, the first two stories explain that as a user, being able to add or remove a pin would help them feel like a part of the community. This tells the

developer that this feature is crucial to the community aspect of the app. Likewise, one of the stories above states that the user wants to filter pins by location to see trees nearby or in an area they want to go to. This helps the developer know what types of filters to add to the map and may inspire new ideas for new features, such as, for example, a new filter to highlight dense areas of trees on the map. User stories allowed us to create a foundation for The Five Elements of UX Design.

4.2.2 The Five Elements of User Experience

In writing User Stories, which developers write from a user perspective, we gained insight into how users would want to interact with the app. These User Stories became the foundation for design decisions as well as reasoning for each feature of the app. Furthermore, they provide future developers and Árboles Mágicos with the capability to understand our thought process and design reasoning. By discussing user needs with our sponsor, our team narrowed down the essential elements necessary to create a crowdsourcing app that induces a sense of community and promotes contribution. Moreover, pinpointing and eliminating unnecessary and excessive functionality from the app resulted in a simpler and less overwhelming final product. From User Story creation, we assessed that there should be five main pages on the app, starting with a login screen, shown below in the form of a mockup made in Figma.

Figure 8: Login Screen Mockup



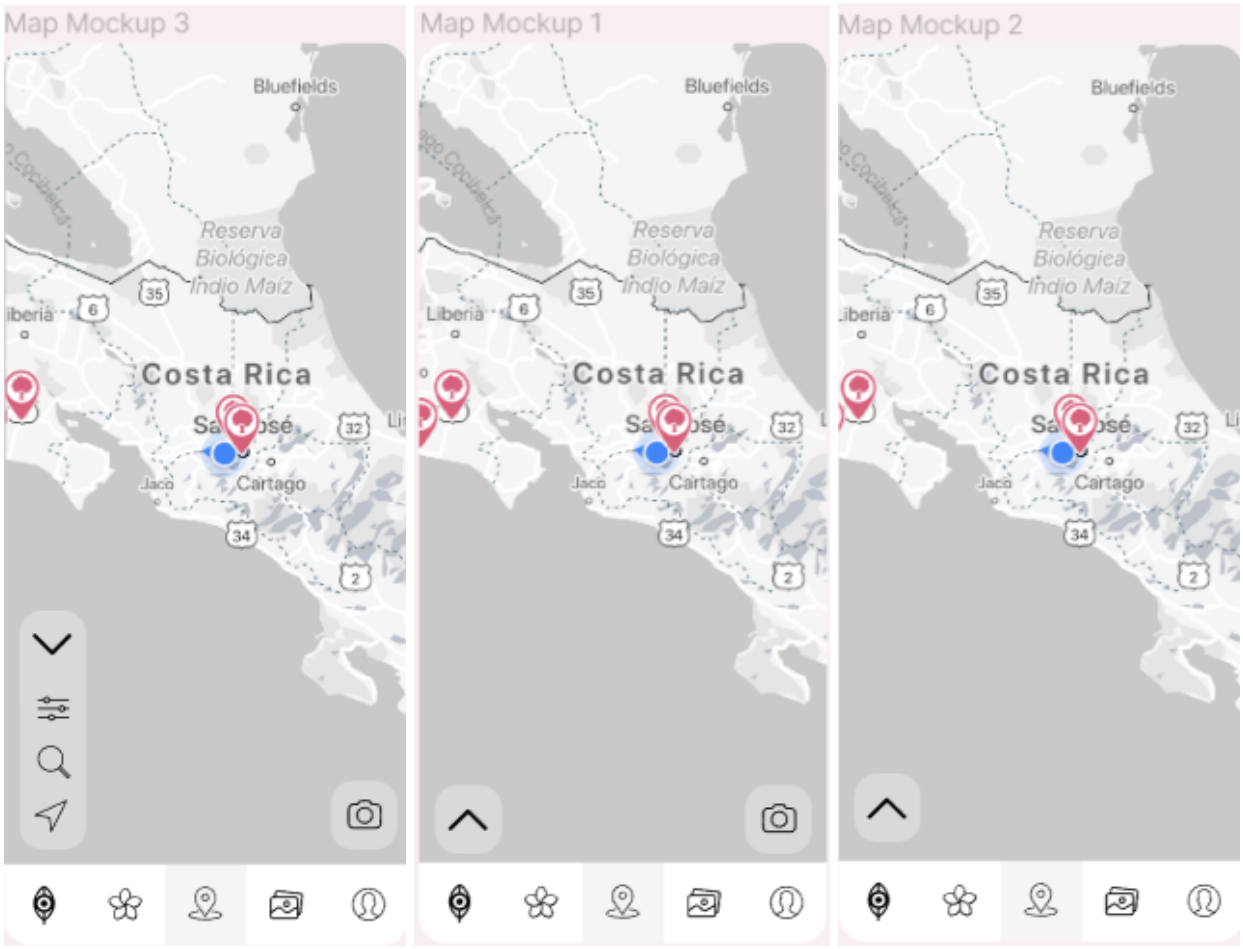
Note. The mockup includes logos and art from Árboles Mágicos that are used and reprinted with permission.

From the login screen, users can sign in to their accounts; however, they can also choose to continue as a guest instead, making the app as inclusive as possible. Additionally, the user can select their preferred language (currently English or Spanish) from the login screen. Through these small features, the app immediately accommodates the user and makes them feel welcome.

The purpose of the login screen is not only to provide a secure way to upload and crowdsource trees to the app but to also give users a sense of being an individual within the app's community who can contribute. One way in which the user can contribute is by posting a photo of a tree and its location to the

map page, which is the first page revealed to the user upon logging in. A mockup of the map page is displayed below.

Figure 9: Map Page Mockup



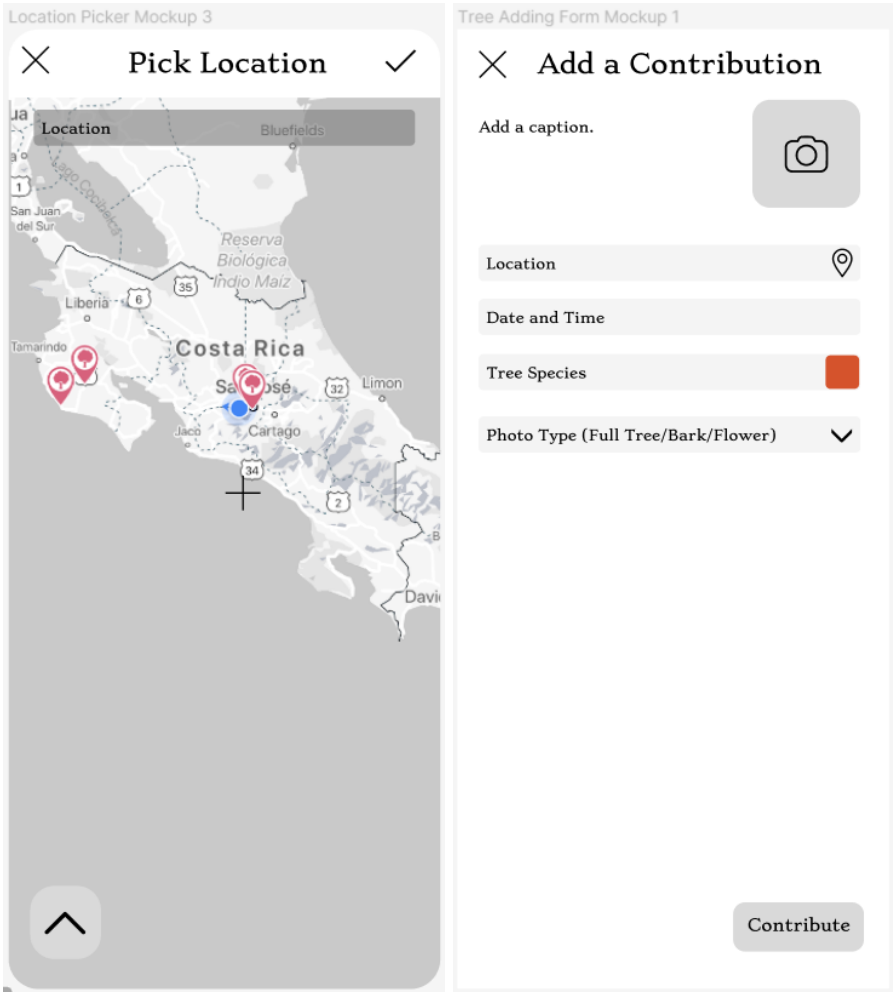
Note. The mockup on the left showcases the navigation features available. The mockup in the middle represents the map page display when the user is logged in and includes an upload button. The mockup on the right shows the map page display when the user continues as a guest, which hides the upload button. Own work.

Due to the map being a crucial crowdsourcing function for the app and the main mechanism for promoting community-based interaction, we decided to highlight it as the main feature a user sees after

logging in. By doing so, users are greeted with a view of all user contributions upon opening the app, which emphasizes their accomplishments. As a result, the user feels the most important and impactful within the app, incentivizing them to contribute.

We wanted the controls for the map to be as simple as possible, so we limited the map display to have two buttons on the bottom left and right corners. The leftmost button, when clicked, opens up a vertical strip menu which contains three buttons: the topmost for filtering, the middle for changing the map view based on a searched location, and the bottommost for changing the map view to center around the user's current location. Through these three actions, users can change the location view of the map and the types of tree pins they see, making the map completely customizable and personalized, especially with the filters. Of the filters, a “Starred by AM” was requested by the organization to highlight and filter the “best” trees on the map. The second button, denoted by the camera icon on the right of the map, brings up the upload form that logged-in users can use to upload records of trees. The mockup for the upload form is shown below.

Figure 10: Upload Form Mockup



Note. The upload form (right) has fields to submit a photo of a tree, its location, the date and time of the photo, the tree species, and the photo type; the location selection screen (left) enables the user to manually type an address or drag the map to pinpoint a coordinate location.

We wanted to make a location selection entry for where a tree photo is taken very flexible. Thus, we made our location selection screen include three options for picking a location. The first option is to search by an address, if the flower is located on public property. The second option is to use the user's current location. The third option is to add a custom location by dragging the crosshairs to anywhere on the map. For now, the upload form is rather simple to make uploads seamless and quick. In particular, we

gave users the option to categorize their photos as a full tree, bark, or a flower, so users can sort through the different types of tree photos taken and not have to view all of them together.

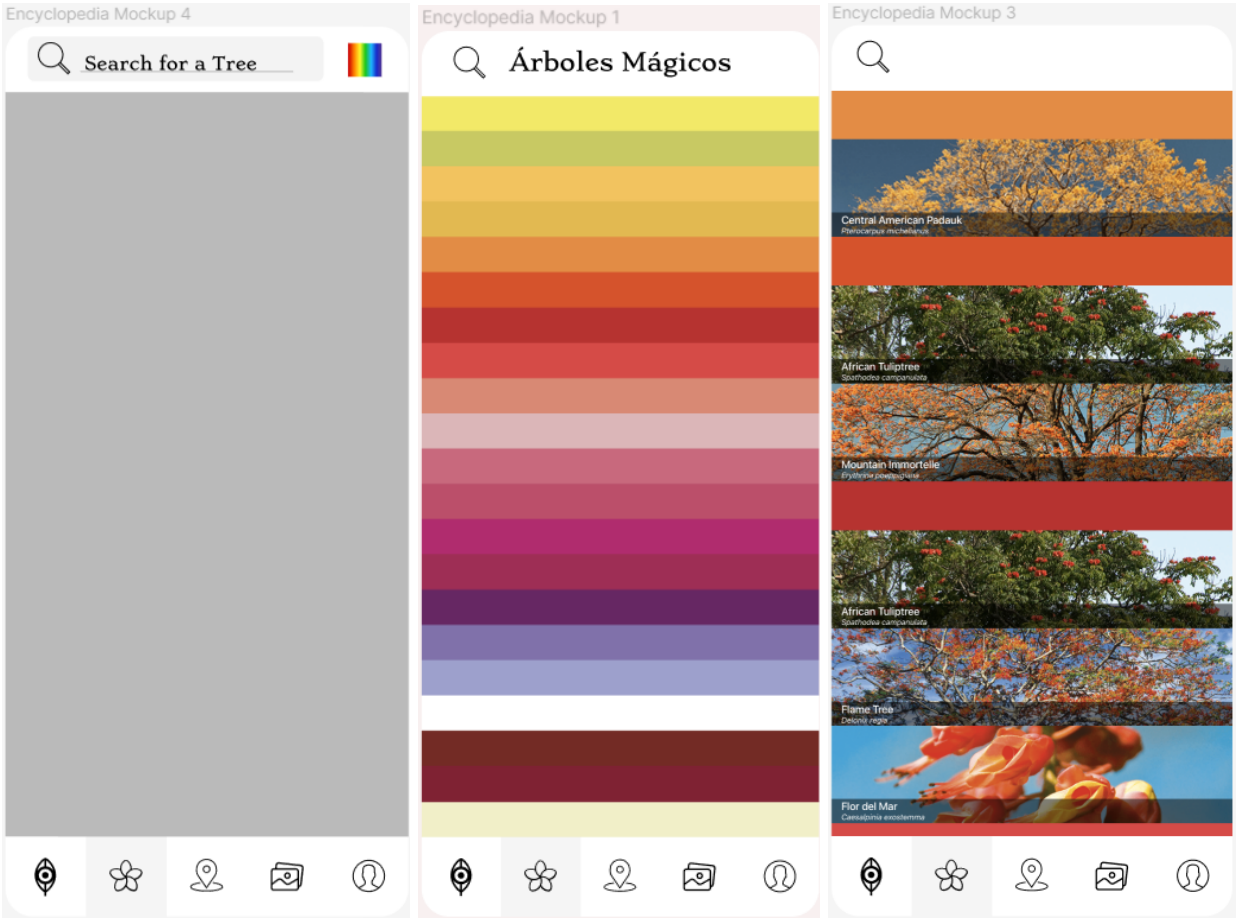
At the bottom of the mockups, there is also a menu bar with five icons, each navigating to one of the five main pages mentioned previously when tapped. All the way to the left of the menu bar, is the icon for the “About Us” page represented below.

Figure 11: Árboles Mágicos About Us Page Mockup

Note. The about us page communicates background information on the Árboles Mágicos organization and their objectives. Own work.

Since the purpose of the app is to build a community by using a crowdsourcing app that embodies the spirit and purpose of Árboles Mágicos, we strongly felt the message of the organization should be made clear via a page solely dedicated to the NGO. This page also features links to the social media of and website of Árboles Mágicos as well. The page to the right of the "About Us" icon is the culture page, displayed below.

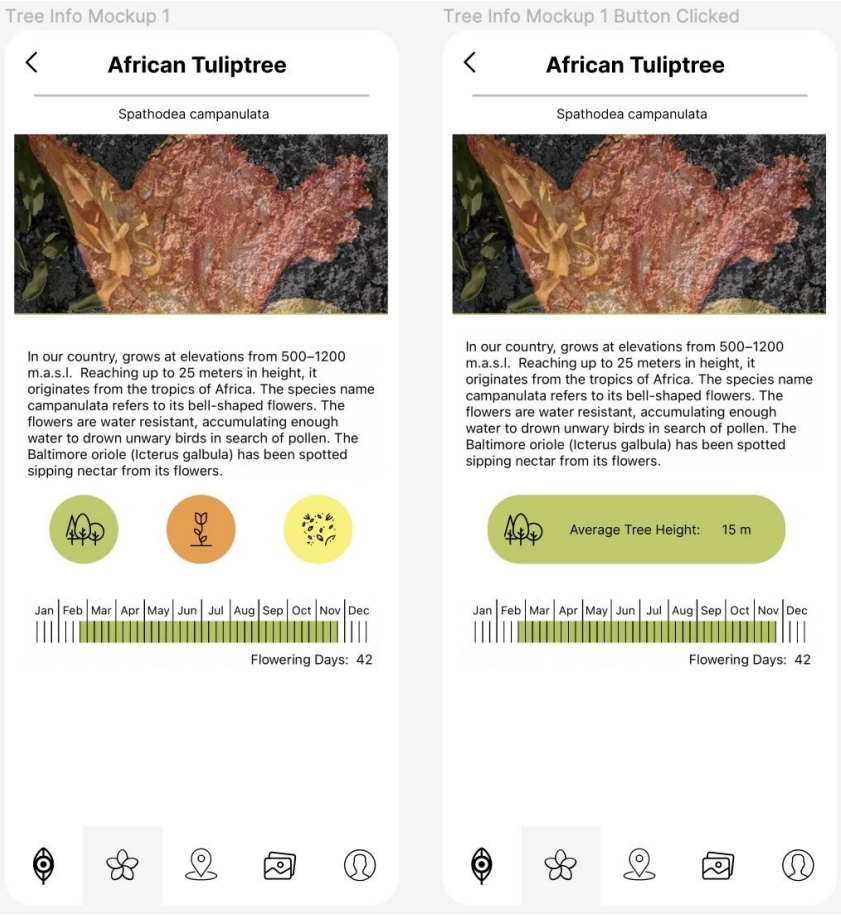
Figure 12: Culture Page Mockup



Note. The middle mockup, the palette, is the initial display of the Culture page upon navigating to it where a user can tap on a color to see trees in that color; the leftmost mockup is when the user clicks the search icon to search for a species and the rightmost mockup is when the user clicks on a color tab to view all the species with that flower shade.

Since the color palette design in the Culture page from the first version of *Ojeadores* was requested by our sponsors, we elected to keep the same design but add a search bar feature to make finding a particular species even easier if the name is known by a user. Furthermore, because the color palette is simple yet visually appealing, especially with its mostly warm color tone, it invites users to click on a color and view different species. When a user chooses a species, the tree information page is displayed. An example of how the page is displayed when the African Tuliptree is tapped is shown below.

Figure 13: Tree Information Page Mockup



Note. The tree information page displays all the tree info, including buttons (first half of the image) that expand to show more information (second half of image). Own work.

The tree information page showcases the name of the tree on the top of the page with the scientific name listed smaller underneath. Previously, tree photos were on a timed carousel view which the user could not control. However, this is not very user friendly, so we implemented a feature to allow the user to manually scroll through the photos or stop a specific image. Along with the photos comes a description of the tree, a timeline of the tree's flowering months, and three buttons that each display different species-specific data when tapped. These three buttons, from left to right, are the average tree height, the average dimensions of the flower, and the method of seed dispersion. Having these buttons

gives a more visually appealing look to the page and condenses the information to avoid overwhelming the user. The last of the main pages featured in the app is the account page.

Figure 14: Account Page Mockup

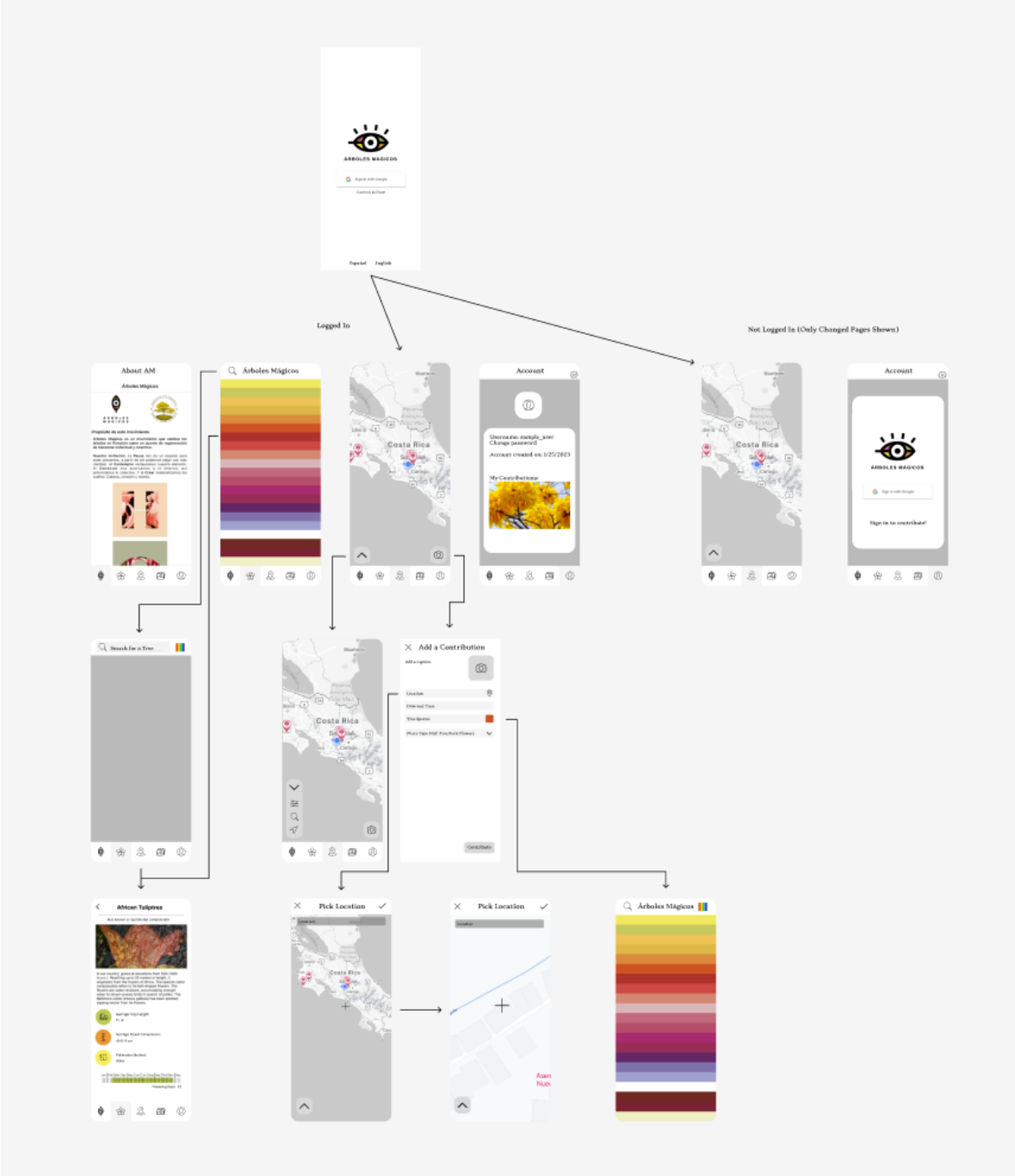
Note. The account page has an icon of the user to add personalization to each account as well as options for changing the user's password and a list of their past contributions.

The purpose of the account page is for managing personal information but more importantly for emphasizing past contributions of a user. A part of trying to make the app more community focused and

active is encouraging users to contribute. If a user sees that they have few or no contributions when viewing the account page, they will likely feel more inclined to add to the tree map to grow their account.

After writing user stories, functionality to meet their criteria, and the justification for each, we mapped out a structural flow diagram of how pages are linked to each other.

Figure 15: Structural Flow Diagram of Page Transitions



Note. Lines with arrows are used to indicate the direction of transition from one page to another; pages in the same row can be centrally accessed from the same place. Own work.

The above general flow diagram was heavily revised based on feedback from our sponsor, and details how all the pages of the app interact with each other. At the top row is the login screen, and in the row below it are two sections displaying the differences in displays depending on whether a user is logged in or not. Four of the main featured pages when logged in, which are mentioned previously, are on the left, and the two pages on the right are when the user is not logged in. The bottom two rows show when specific buttons are tapped throughout the app and what pages they lead to. Our purpose in generating the diagram was to ensure no subpages are buried too deep within one of the five main pages and that pages do not navigate to each other in a cyclical manner.

4.2.3 Logistics of Crowdsourcing

The implementation of crowdsourcing features into the app was an important aspect of the app's overall design. Leveraging the collective skills of users on an individual basis is a critical factor in ensuring the effective, efficient, and secure operation of the app. While public testing was not done over the course of this project, it can be a useful resource when the app is published, as it allows future developers to know what does not work and what can be improved upon.

One such feature was implementation of community flagging, which lets users mark posts or photos as potentially incorrect or inappropriate. Sometimes a user incorrectly identifies a tree species, and this allows users to mark those posts as potentially incorrect so they are brought to the attention of administrators, as users are not able to edit another person’s post.

In addition to the flagging of incorrect information, allowing users to flag posts for inappropriate content is a very important feature to make crowdsourced. Since there is freedom to post photos from a user’s camera roll or directly taken through the app, some users may upload photos that are not tree-related. This can undermine the goals and positive nature of the app. While we utilize the Cloudmersive API that checks a photo before it is posted to ensure it is appropriate, some photos may still make it through to the map. Any posts flagged for inappropriate content are immediately hidden on the

app, and they are brought to the attention of the moderators. Moderation is a major part of making a community safe and family-friendly, especially when users are given the ability to post and like photos in the app. Giving users the ability to contribute to moderation not only helps the community but also the team of engineers tasked with maintaining and monitoring the app. With users being able to flag content, the moderator's workload is reduced. Instead of going through every photo posted and having to correct or take down posts, the moderators only have to go through the flagged posts, allowing corrections to be made quicker. This feature not only reduces the workload for administrators but also fosters a sense of community engagement and ownership among users, ensuring that the app maintains a safe and positive environment for all users.

Being able to draw on the community for decision-making keeps the app running smoothly and safely. This is due to the fact that having many users fact-checking information is more effective, fast, and efficient when compared to the scenario in which all of that work falls on the administrators. Furthermore, crowdsourcing also promotes a sense of community engagement and ownership among users. Users feel valued and invested in the app as they are given a platform to contribute to its success. This sense of community and collaboration helps to foster a positive and supportive environment for users, promoting the app's goals of community connection, citizen science, and individual reflection.

After an analysis on crowdsourcing within the app, the benefits of crowdsourcing tasks in the Árboles Mágicos app are significant. Crowdsourcing helps to reduce the workload for administrators, promotes a sense of community engagement and ownership among users, and ensures the accuracy and reliability of the information shared on the app. These benefits make crowdsourcing features in the Árboles Mágicos app an important aspect of the app's overall design, helping to ensure its success and continued engagement of its users.

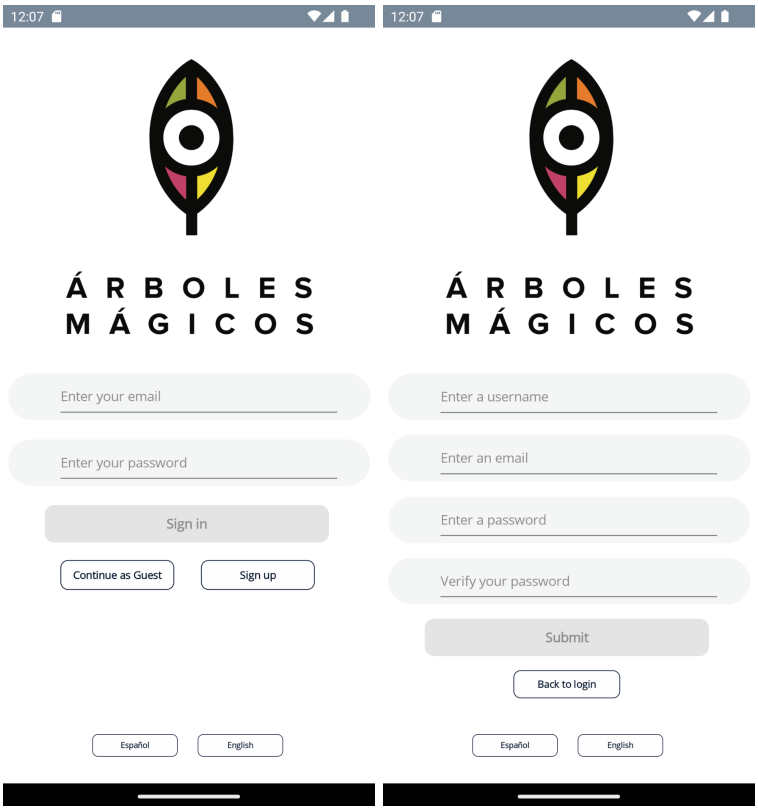
4.3 Final Android Application

Over the course of a month, we developed an android version of the previous app owned by Árboles Mágicos. Not only did the new app take most of the features from the previous app, but also added a large array of new features, designs, and pages that create a larger platform for users to pause, connect, contemplate, and create.

4.3.1 New and Redesigned Pages

The pages of the new Android application offer a similar feel to the previous app, however, with some changes. Previously the *Ojeadores* app consisted of five pages, each listed in the bottom bar of the screen: a culture page, a "Search by Region" page, the color palette page, a "Search by Name" page, and finally an "About Us" page. In the new application, the five icons in the bottom bar navigate to the new, from left to right, the "About Us" page, the Culture page which includes the color palette and the search bar, a new map page, an events page, and finally the account page. In addition to these five main pages, we created a Login page when the app opens to allow the user to sign in. Also included in the previous app was the tree information page which has been completely redesigned.

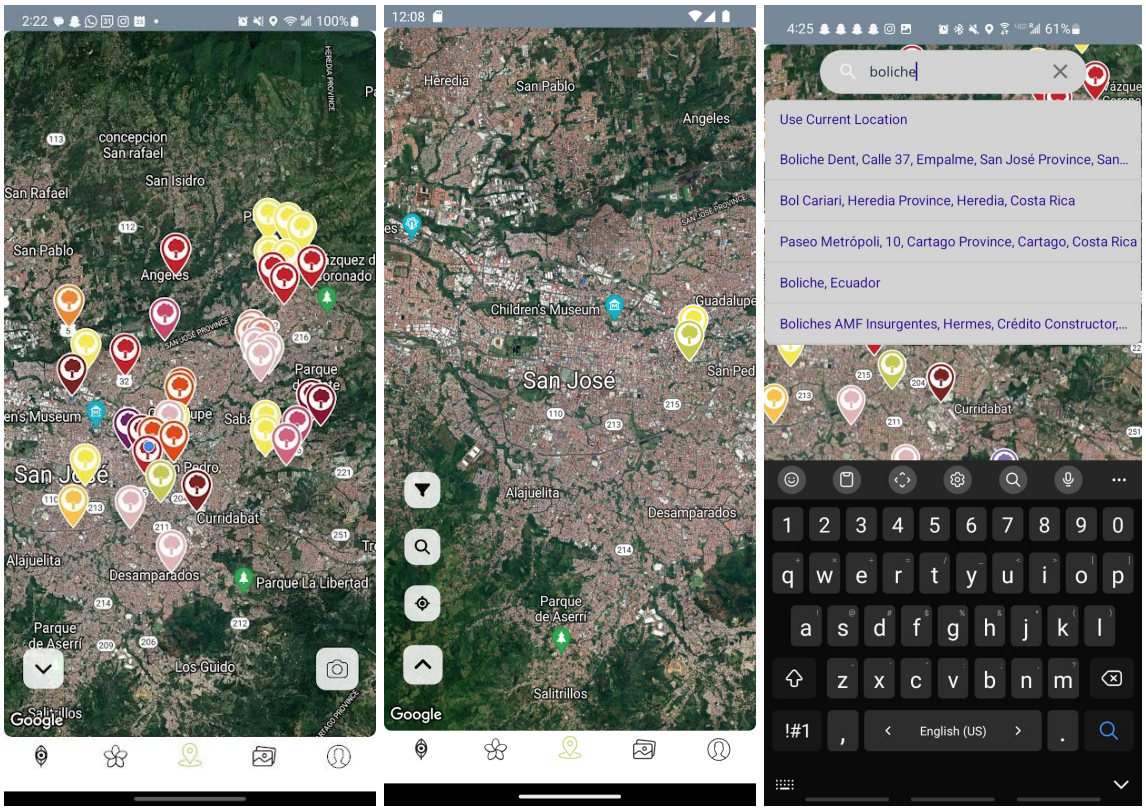
Figure 16: The Login Page



Note. The first screenshot shows the login page when a user can sign in using their email/username and password. The second screenshot shows the login page after the sign up button was tapped and allows the user to create an account.

We created an account system to be able to increase the sociability of the app. The login page is the first page that is displayed when the user opens the app. From the page the user can sign in with their email or username and password or create a new account. Once the user has an account, their data is saved in the database, and the app can auto-login every time the app is opened. The user does not have to login or create an account as they can tap the “Continue as Guest” button next to the “Sign up” button. This page also includes buttons to switch the app’s language between English and Spanish. After the user logs in or continues as a guest, they are brought to the map page.

Figure 17: The Map Page



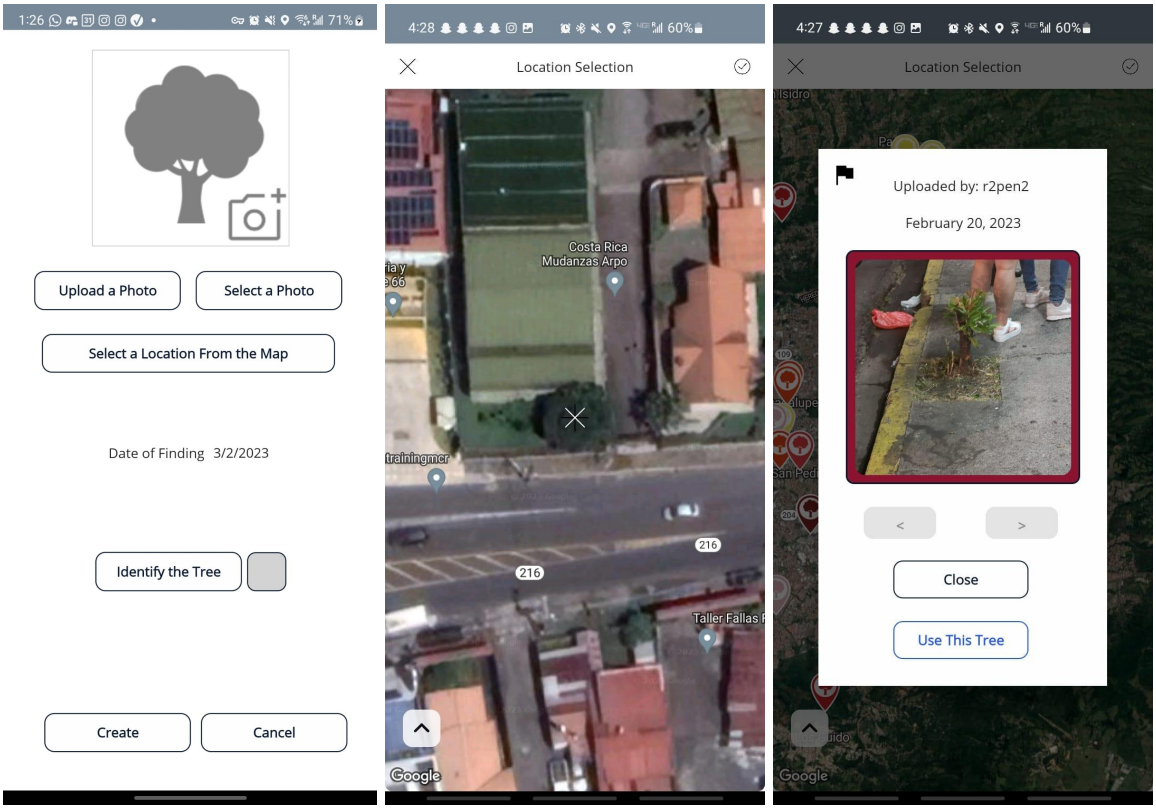
Note. The first screenshot shows the map page with pins. The second screenshot shows the map menu expanded after the arrow was tapped. The third screenshot shows the search location feature that also auto completes with known locations from Google. Own work.

The map page is one of the new features added to the app. It allows users to see flowering trees in their location while also allowing them to add new trees to the map or pictures of existing trees to those pins. This page offers a gateway to many of the new features of the app. If a user taps on the arrow on the left, they are given three options: a map filter, a location search, and a button for centering the map on the user's current location. The filter page shows a new page that allows the user to filter by the species name or color and includes a button to remove all applied filters. The search location option allows for users to search for landmarks or businesses as well as a specific address to see trees nearby. Finally, the button for centering the map on the user changes the map view to the user's current location. In addition to the arrow

button, there is a camera button on the bottom right of the page that brings the user to the upload page. This button is not shown if the user is not logged in.

The upload page allows the user to create a new pin or add to an existing pin. The page includes both a way to upload an existing photo and take a photo within the app. After uploading a photo, a user can choose the date the photo was taken, the tree species, and the color of the tree from the color palette. The tree species is optional due to the possibility that users do not know which species the tree is. The user can also choose the location with the location picker.

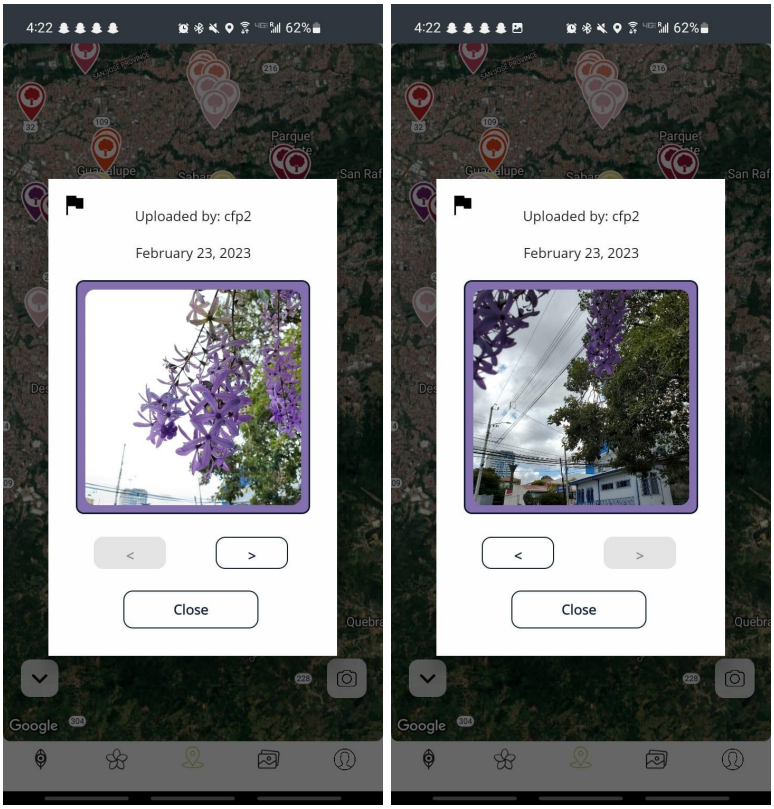
Figure 18: The Upload Page



Note. The first screenshot displays the upload page where the user can upload a tree to the map. The second screenshot shows the location picker with crosshairs to select a precise spot. The third screenshot shows selecting a pin from the location picker.

The location picker defaults to the user's current location. However, they can drag the map to a different place or search for a specific location. Here the user has two options: use an existing pin or create a new pin. If the user wants to use an existing pin, they can bring the crosshairs over that pin and select it. They are then given the option to "Use This Tree". If the user wants to create a new pin, they can select a new, unoccupied location on the map and the new pin will be created. After everything necessary in the upload page is filled out, the user can submit the post and it will immediately be added to the map.

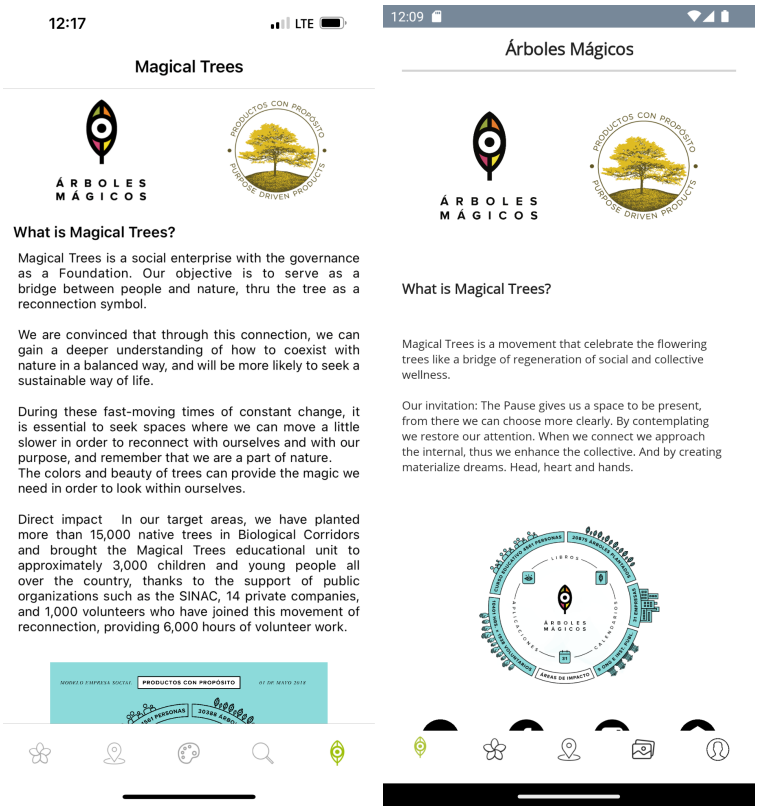
Figure 19: The Tree Popup



Note. The tree popup shows photos from a pin as well as the author's username and the date it was posted. Own work.

From the map page, a user can view others' posts simply by tapping on a pin. When a pin is tapped on, the tree popup page is displayed. As shown above, the popup page consists of the uploader's username, the date the photo was uploaded, the photo or photos uploaded as well as arrows to navigate between photos, a flag button, and a close button to exit the popup. The flag button is part of the application moderation and allows for users to send the post to the administrator portal to be reviewed.

Figure 20: The "About Us" Page

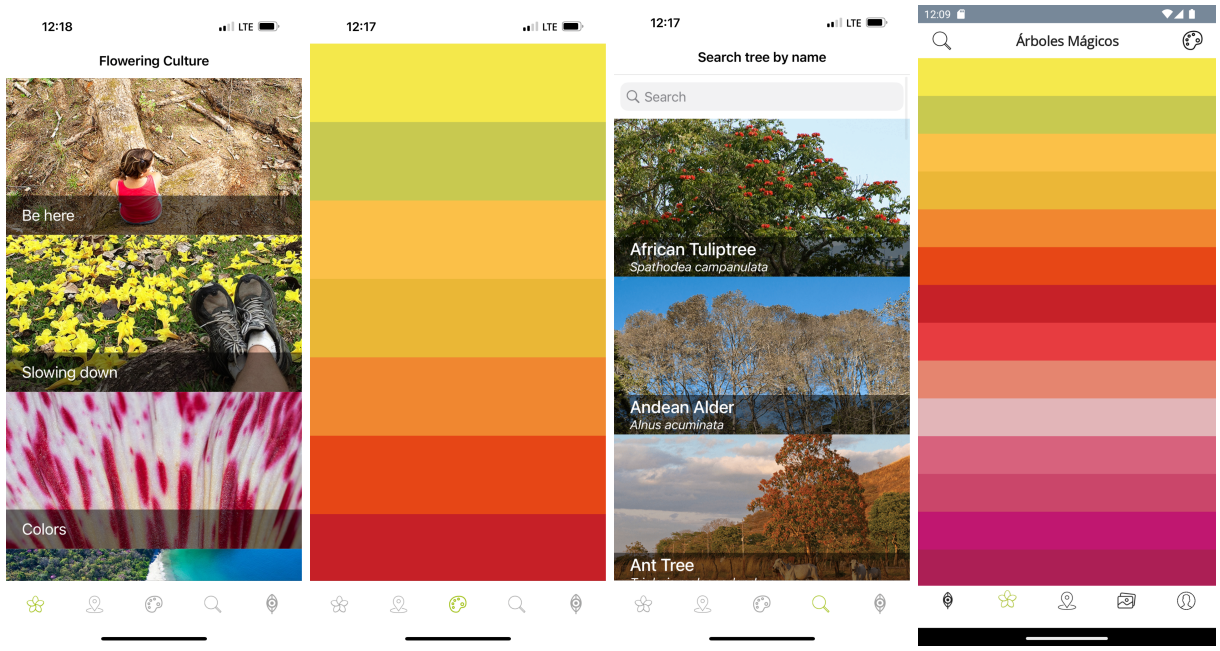


Note. First image: The old design of the "About Us" page displays information about Árboles Mágicos as well as links to their website and social media. From *Version 1.0* [Digital Image], by Árboles Mágicos, 2013. Reprinted with permission.

Note. Second image: The new design displays the same information as before but with correct translations and without the language button. Own work.

The “About Us” page functions very similar to the previous app’s page. However, we have implemented a cleaner look. On both the current and previous pages, the Árboles Mágicos logo and the Purpose Driven Products logo are static on the top of the page. Then there is a description of the mission. In the previous app, the English translation did not make sense, so we wrote a new translation from the Spanish version. There are four buttons that redirect the user to the website, Facebook, Instagram, and email of Árboles Mágicos. On the previous application, there was a button to switch the app’s language. We relocated this button to the Login page.

Figure 21: The Culture Page

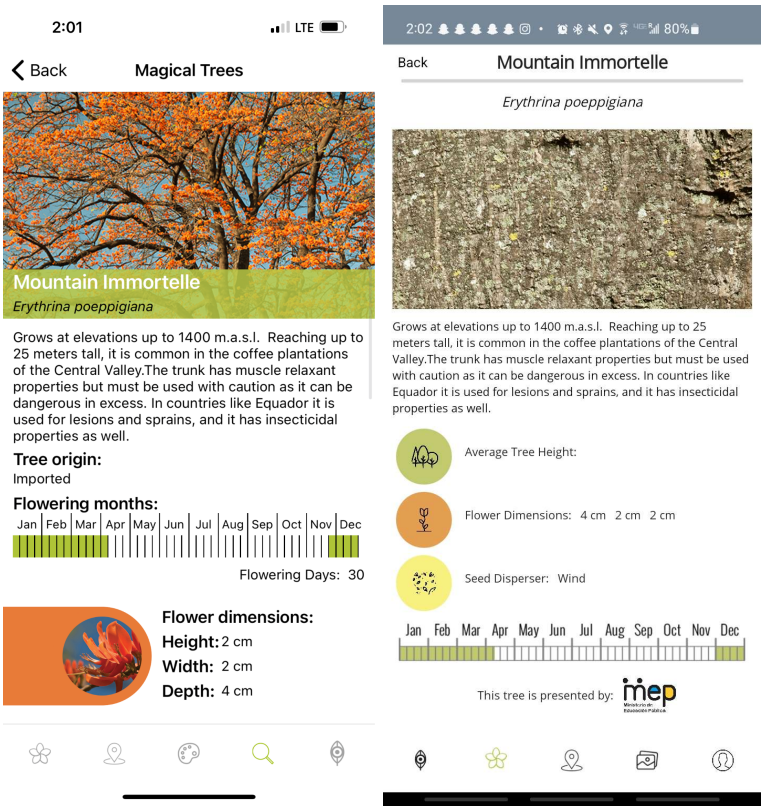


Note. First three screenshots: From left to right, the culture page, the color palette page, and the search page. The culture page displayed information about the pictures shown. The color palette allows a user to select a tree from a color. The search page allows a user to select a tree from its name. From *Version 1.0* [Digital Image], by Árboles Mágicos, 2013. Reprinted with permission.

Note. Fourth screenshot: The new culture page combines the functionality of the three previous screenshots. Own work.

The culture page previously consisted of four photos that dropped down and displayed a small paragraph about each photo, such as a meditation prompt. After careful consideration, we decided to remove this future completely, as it had a lot of bugs and did not contribute to the app's purpose. We reinvented the culture page to include the color palette and a search bar, so users can find trees and learn more about the species. The search icon at the top of the page expands to the search bar, while the palette icon brings the user back to the color palette. The user can tap on a color and expand a list of trees with that color, similar to the previous app. When a tree is tapped, the tree information page is displayed.

Figure 22: The Tree Information Page

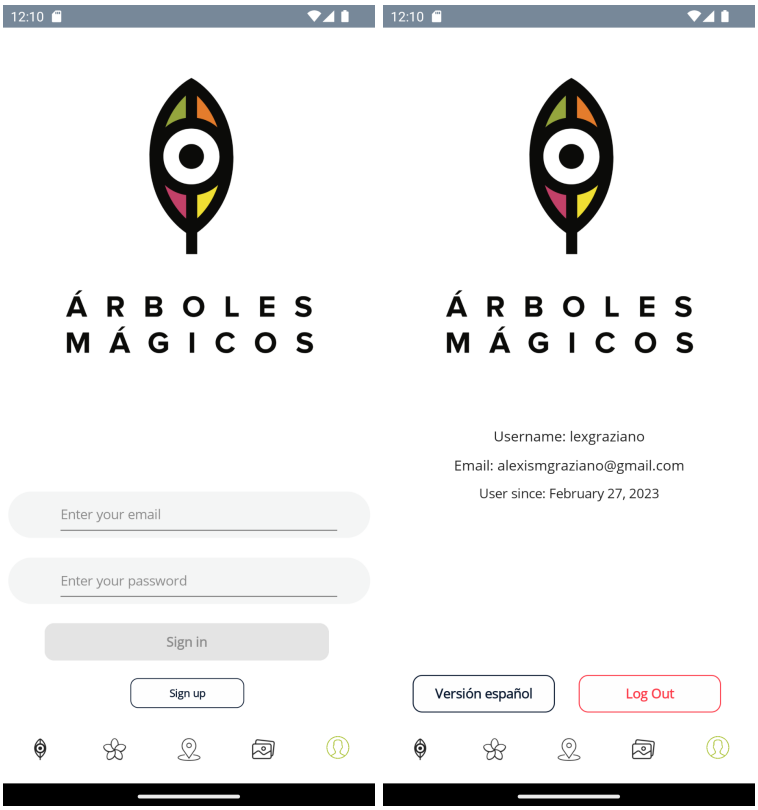


Note. First Screenshot: The tree information page displays all the tree information according to species. From *Version 1.0* [Digital Image], by Árboles Mágicos, 2013. Reprinted with permission.

Note. Second screenshot: Displays all the tree information but with a cleaner design. Own work.

The tree information page serves the same purpose as it did in the previous app. However, we created a new design that not only catches the user's attention, but also makes the screen less cluttered while displaying all the same information. The first screenshot came from the previous app and includes a scroll function which hides a lot of information. The second screenshot shows the new design and includes all the information without the need of the scroll function. We removed the Árboles Mágicos name from the top of the screen and replaced it with the tree's name and scientific name. Previously, the photos played on a timer; however, now the user can scroll through the photos. The average tree height, flower dimensions, and seed disperser layout in the *Ojeadores* app was clunky, so we created a design that is more pleasing to the eye. We also added a flower icon to replace the flower photo, making the design more uniform. We also rebuilt the blooming calendar since the previous one had many issues.

Figure 23: The Account Page



Note. The first screenshot shows the account page when the user is not signed in and looks similar to the login page. The second screenshot is shown when the user is signed in and displays the user's data as well as an option to sign out. Own work.

The account page allows the user to sign in and out as well as change the app's language. If the user is not signed in, the account page functions very similarly to the login page where the user can sign in or create an account. Once signed in, the account page displays the username and email associated with the account and date the user was created. From here, the user can also sign out or switch languages.

The events page only consists of the "Events" title on the top of the page. This feature was not completed due to a lack of time. However, this page would display notifications that administrators would deploy about upcoming events, new trees found in the area, and possibly any news Árboles Mágicos would desire to share with their app's users.

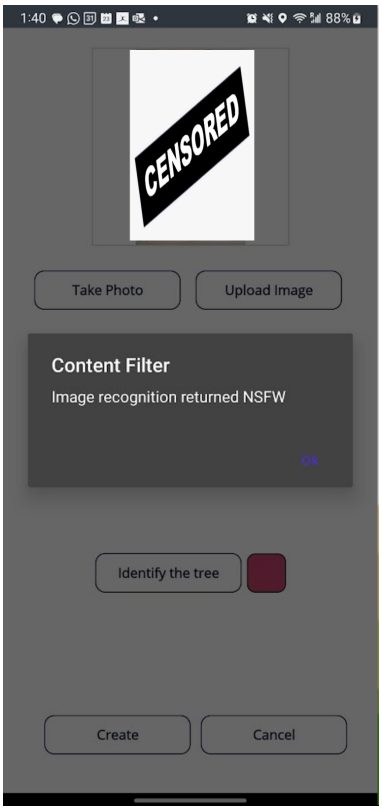
4.3.2 Moderation

There is a standalone moderation portal built in React.js to allow Árboles Mágicos to ensure that all content on the app is accurate and inoffensive. Users have the ability to click a flag icon on any photo, which will place the photo in a section of the moderation panel for "flagged" contributions. Any moderator (likely Árboles Mágicos forest engineers or trusted community members) are then given the option to either delete or "verify" the photo. Verifying the photo removes the flag and disallows users from flagging it in the future, while deleting the photo simply removes it from the map. Anyone with access to the moderation portal also has the ability to ban any user, which removes all of their contributions from the map and blocks them from uploading anything in the future.

Árboles Mágicos suggested a few changes to how moderation works, but we did not have time to implement them. Firstly, they would like the ability to create "events" that appear on the map with special marker icons. These events would be manually published by Árboles Mágicos employees through the portal. They would also like to have a hierarchy of moderators with different permission levels. At present, anyone with access to the portal has the same power to remove posts and ban users. Ideally Árboles Mágicos could entrust members of the community to delete flagged photos, without allowing them to create events or ban others from using the app.

We also discussed how users may moderate themselves without involving Árboles Mágicos directly. For issues with contributions that aren't as pressing, such as a tree with the wrong color or species listed, users could submit anonymous notes to the original poster. The original poster would then have some time to resolve the contribution on their own. Allowing users this ability would not only take stress off of the Árboles Mágicos moderation team but would also strengthen the community by encouraging positive and constructive interaction between users rather than reporting each other to the organization.

Figure 24: Censorship



Note. This screenshot shows an example of a photo being denied from uploading onto the map due to it not being appropriate.

To minimize the possibility of any users uploading inappropriate content to the map, we implemented Cloudmersive's Image Filtering API. Before user generated content is submitted to the database, we send it to Cloudmersive to determine the likelihood that the photo may be NSFW. If the API suggests that there may be a risk in allowing the photo to be displayed on the map, the user is blocked from uploading it. Cloudmersive allows 800 API calls every two months, which was enough for development but will likely not be enough to filter all submissions once the app is released (Cloudmersive, n. d.-b). Cloudmersive pricing information was included in the developer manual and in section 4.5 of this paper (Scalability).

A way to reduce the clutter of posts is to limit how many photos a user can add to a tree. After careful consideration of the different types of photos possible for each tree, a three-photo limit on each tree allows for users to be able to post photos of the flowers, the leaves, and the tree in one post while also giving the freedom to post random combinations of types of photos.

4.4 Operations and Maintenance Manual

The creation of a maintenance manual for the Arboles Magicos app satisfied the need to ensure that Árboles Mágicos can effectively maintain the application in the future and keep it running smoothly, while easily allowing an outside programmer to pick up the project where we left off, to update it as necessary. As such, there are two sections: a guide for Árboles Mágicos and a guide for future developers.

The first Árboles Mágicos section of the manual provides instructions and guidance on updating global content, moderating user-loaded content, as well as other common tasks. This manual is in document format with pictures and visual instructions, in both English and Spanish. Uploading new events is going to be crucial for building a community for the app. The manual will explain how to create notifications to alert all users about events, stories, or new pins. Knowing how moderation works is very important, since it determines which users can use the app. The manual will contain a guide on how to remove posts, ban users, and mediate comments. By providing this information, we can ensure that anyone tasked with monitoring the app will have no problems. In the case that there is an issue that the administrators cannot fix, our contact information will be attached.

The developer-oriented section to the maintenance manual contains information on how the application was built, as well as design choices and considerations to keep in mind when making changes to the underlying application. This is contained in the README.md in the Git repository that contains the code for the application. The README.md is a text file with documentation on the codebase such as the

contributors, the tools used to create the app, the external libraries used, the code and class structure, and the database structure. Along with that, instructions on how to run and debug the application are included to ensure that someone can quickly pick up the project in the future.

The maintenance manual is an essential tool for ensuring the longevity and effectiveness of the Arboles Magicos app, and it provides peace of mind for both the organization and its users. We hope that this effective and detailed documentation will contribute positively to the maintenance of the app and future updates.

Alongside the maintenance manual will be a user guide. The user guide will be a simple How-To tutorial built into the app when the users first open after installing. The tutorial will show what each button on the bottom bar does and walk the user through uploading a photo through the map page. This guide will ensure that all users can use the app to its fullest potential and dissipate any frustration caused from not understanding how to use a specific feature. The user guide will also be accessible in the settings page under the account page.

4.5 Scalability

During development, we used the free versions of Google Firebase and Cloudmersive API, which were services we used for storing and retrieving data as well as checking for inappropriate photo content. These free plans were sufficient for development and testing purposes, but as the app gains more users, Árboles Mágicos will likely have to invest in paid plans to avoid exceeding usage limits.

The Firebase free tier should be enough for the near future, but we expect that Árboles Mágicos will have to upgrade their Cloudmersive plan if they would like to continue filtering content automatically.

Figure 25: Firebase's "Firestore" Database prices as of 3/2/2023

Quota	Free Until	Then
GB Stored	1GB	$0.18/GB/month
Document Writes	20,000 uses/day	$0.18/100,000 uses
Document Reads	50,000 uses/day	$0.06/100,000 uses
Document Deletes	20,000 uses/day	$0.02/100,000 uses

Note. Prices are from Firebase's official pricing page. These prices represent the cost of interacting with a database created from Firebase. *Writing*, *reading*, and *deleting* are all actions performed on data within a database and the costs associated with them are based on how many times those actions are performed. After the number of uses shown in the middle column is exceeded, the pricing from the last column is applied. Data from Firebase, n.d.-b.

Currently, based on our interactions with the database through constant development and testing, we noticed that exceeding the numbers shown in the "Free Until" column are rather difficult. Even though we were a group of four individuals developing the app, we had to upload new tree data more often than the typical user would need to to fully test the functionality of the app. Thus, we concluded that, at least for the early release of the app, exceeding the limits of the free plan for Firebase will not be an issue. Moreover, developers can further reduce database usage via better development methods.

Figure 26: Firebase's File Storage prices as of 3/2/2023

Quota	Free Until	Then
GB Stored	5GB	$0.0026/GB
GB Downloaded	1GB/day	$0.12/GB
File Uploads	20,000/day	$0.05/10,000
File Downloads	50,000/day	$0.0004/10,000

Note. Prices are from Firebase's website. These prices are associated with the cost of storing and retrieving image data only as opposed to tree data. Data from Firebase, n.d.-b.

From a technical standpoint, uploading and downloading images using the free plan from Firebase will not be an issue in the long term, as the size of image data can be reduced through development techniques. However, whether these costs will be exceeded or not will be determined upon release of the app since user testing is crucial for analyzing cost.

Figure 27: Firebase's User Authentication Service prices as of 3/2/2023

Monthly Active Users (MAU)	Price per MAU
0 - 49,999	$0
50,000 - 99,999	$0.0055
100,000 - 999,999	$0.0046
1,000,000 - 9,999,999	$0.0032
10,000,000 +	$0.0025

Note. Prices are from Firebase's website. Data from Firebase, n.d.-b.

The cost of authenticating users when they log in to the app is relatively inexpensive. Therefore, these costs should not be taken into account as severely as the cost of other aspects of the app. However, they should still be noted.

Figure 28: Cloudmersive Image API's prices as of 3/2/2023

Quota vs. Price	$0.00/month	$19.99/month	$49.99/month	etc.
API Calls	800/month^2	20,000/month^2	50,000/month^2	See Pricing
API Call Frequency	1 call/second	2 calls/second	2 calls/second	See Pricing
Max File Size	± 3.5MB	± 1GB	± 4GB	See Pricing

Note. Prices are from Cloudmersive's website. Month squared means bimonthly. The table represents the different tiers of pricing that Cloudmersive offers and what they include. Data from Cloudmersive, n.d.-b.

Because image content safety is a huge concern and priority, we have concluded that the free plan is not sufficient for the intents and purposes of the app. The first major issue with the free plan is the max

file size. Based on testing image upload with the app, we found that the file size limits dictated by Cloudmersive are tolerable for smaller photos taken with a mobile phone. Yet, for professional photos with a higher resolution and picture quality, the file size limit is easily exceeded, which is why Árboles Mágicos should purchase a higher tier plan. Still, further testing and analysis is required for assessing how to handle larger file sizes and which exact plan is necessary.

Chapter 5: Conclusions and Recommendations

5.1 The Importance of Árboles Mágicos

Árboles Mágicos aims to foster a healthy connection between modern life and nature through flowering trees. The organization conducts tree planting sessions, professional development workshops, nature walks, and classroom workshops to bring the community closer to this goal. Together with the crowdsourcing networking abilities of the Árboles Mágicos app, the organization aims to have a more widespread impact on the Costa Rican community.

5.2 App Development Process

When building the Android application deliverable, to ensure the development process of the app incorporated as much feedback as possible and involved justification for every feature added, we used two well-known software engineering methodologies of design: Agile Scrum and The Five Elements of User Experience. Agile Scrum is a framework for developing software quickly and effectively by enforcing concepts of iterative design. To follow Agile Scrum, we planned a list of weekly goals and features to accomplish for the app at the start of every week, ranked the priority of each, and then assigned tasks to each member of the team. After each week, we gathered feedback from our sponsor and reviewed the features we developed to determine how they could be improved or how they should change. Thus, we were able to stay productive and on track with finishing the core app features by the end of the seven weeks.

Since building an app is a complicated process and time consuming, it was important to carefully plan out which features were essential to have and to back those decisions with written justification. Therefore, we used The Five Elements of User Experience template, which breaks down the application

development process into five stages of design that range from abstract to concrete. Writing out a thorough design plan not only helped us better prioritize our time but also benefited Árboles Mágicos in gaining a better understanding of why we made certain design decisions. Furthermore, the written justification we wrote will also help future developers understand our thought process.

After following the first two stages of The Five Elements of User Experience, where we planned the overall features of the app on paper, we used Figma, a website for designing mockups, to create visual representations of how each page of the app might appear, including a flow diagram to show pages will transition between each other.

By presenting these mockups to our sponsor, we gathered feedback but most importantly began to discuss the logistics of crowdsourcing. A requirement related to crowdsourcing from Árboles Mágicos was to make the app community-centric, yet not a social media app. To achieve this, we focused on creating a rating system that is hidden from the user to eliminate competition between users, since this type of competition detracts from the app's purpose of community-building around the appreciation for flowering trees. User contribution is encouraged through community moderation, where one can point out incorrectly labeled species and pick aesthetically-pleasing photos. To achieve this, we designed a hidden, anonymous rating system where users can downvote or upvote a photo to help indicate whether the photo is of bad or good quality. Furthermore, users can flag photos for being inappropriate.

After finishing the development of the app, we wrote a brief and concise operations and maintenance manual. The manual details our design decisions, resources used, code structure, success and failures, and directions for maintaining the app in the future.

5.3 Deliverables

5.3.1 Android Mobile Application

After taking the database of trees (including colors, size, scientific name, blooming times, etc.) from the previous app, *Ojeadores*, we constructed a new app with new features allowing more user connection and creating easy-to-read displays for each aspect of the app. The most notable of these features is the addition of a map where users can post pictures of their tree findings and others can view and add to posts in real time. This is a major part of fostering connection since in the previous app, there was no user to user experience available. Another feature we have added is the events tab that allows users to see new posts from administrators, upcoming events hosted by Árboles Mágicos, and more. This will allow people to not only connect through the app, but in-person at these gatherings to find trees and explore the beauty of nature. On top of these features, we have cleaned up the app to have new designs to improve user experience and fixed any bugs associated with the previous app.

Once the mobile application was completed, we transferred ownership of the codebase to Árboles Mágicos on GitHub. This lets the organization continue development on the application using GitHub's code-sharing features even after the project has concluded.

5.3.2 Developer Maintenance Manual

This section of the manual is a crucial part of the development process as it details how to change and build upon the code written for the app. In the code delivered to Árboles Mágicos exists a document titled "README.md". The README file contains several short chapters that explain how the more complex aspects of the codebase work. The developer maintenance manual covers the following topics: pushing and fetching database objects, updating the map object, how the pages are laid out, known issues, common errors and their solutions, how to make changes to the administrator portal, pricing for any tools used, and an appendix with miscellaneous notes.

This manual, stored in the GitHub repository along with the code, is not intended to be viewed by anyone outside of Árboles Mágicos. The repository is private, but out of an abundance of caution, no sensitive information has been included. All code secrets—database keys, github access, web-hosting, etc— must be obtained by speaking to Árboles Mágicos directly. We have also included our names and emails in the event that someone has trouble getting necessary information from the organization.

5.4 Recommendations

When our completed work is handed off to Árboles Mágicos, the organization needs to maintain its functionality and usefulness. We have a number of recommendations for the organization moving forward, as well as recommended actions if they would like an expanded feature set in the future.

While the Android application that we developed is functional and visually styled, there is room for growth in its list of features, as we were not able to implement the entire feature set desired by the foundation in the time we had for the project. The major reason as to why was due to our lack of prior experience with the .NET MAUI framework. Therefore, we recommend that Árboles Mágicos employ a developer with prior knowledge of the .NET MAUI framework in the future, which will reduce the time spent learning the software and in turn hasten the development process of more features. Some features, which we could not fulfill in time but are important to the future success of the app are an event system, a built-in calendar with bloom periods, more community moderation tools, and a color blind mode. Among these features, we consider the event system to be the most significant feature currently not in the app due to its potential to strengthen the community and heighten the crowdsourcing aspect of the app. Moreover, by increasing the moderation capabilities of the community, it will further promote user contribution within the app while also decreasing the workload for the Árboles Mágicos admins and moderators.

On top of an extended set of features, Árboles Mágicos would like to release a similar application to the App Store for deployment on iPhones. The app was developed for Android after taking into account that the majority of the organization's original user base uses the Android operating system, but it was built on a framework that makes it relatively easy to port the app to iOS.

To maintain the application, the organization requested special administrator accounts with the ability to moderate the app and promote specific Árboles Mágicos community events. We recommend that Árboles Mágicos appoint someone in the organization to lead post moderation and community event posting. If the app traffic grows enough that one person is not enough to moderate everything, multiple people should be appointed to assist in the moderation efforts.

5.5 Conclusion

With this next version of the app, Árboles Mágicos hopes to further their vision of spreading appreciation for flowering trees by developing a mutually beneficial relationship between people and the environment. The two deliverables for this project were the mobile app, remade for Android with new features and updates, as well as the maintenance manual to help the organization and future developers. Through the app, Árboles Mágicos has the potential to reinvigorate appreciation for the beautiful flowering trees that local Costa Ricans see in their day-to-day life while being a platform for scientific collaboration. The app lets both normal people and scientists upload records and photos of trees, creating a map of both beautiful photos and scientific data. We hope that Árboles Mágicos will continue to invest time into this application and use it to expand their organization's reach and impact.

Works Cited

[About Árboles Mágicos]. (n.d.). Árboles Mágicos. https://arbolesmagicos.org

About Costa Rica: Embajada de Costa Rica en dc. About Costa Rica | Embajada de Costa Rica en DC. (n.d.). Retrieved November 18, 2022, from http://www.costarica-embassy.org/index.php?q=node%2F19#:~:text=Costa%20Rica%20receives%20over%201.7,do%20eco%2Dtourism%20related%20activities

Amazon. (n.d.-a). *Getting started with Ios app development*. Amazon Web Services, Inc. https://aws.amazon.com/mobile/mobile-application-development/native/ios/

Amazon. (n.d.-b). *What is an API? - Application Programming Interfaces Explained - AWS*. Amazon Web Services, Inc. Retrieved March 1, 2023, from https://aws.amazon.com/what-is/api/

Appixi. (2022). *LeafSnap - plant identification* (Version 2.33) [Mobile app]. Apple Store. https://apps.apple.com/app/id1487972880

Árboles Mágicos. (2020). *Version 2.0* [Digital Image].

Bonney, R., Phillips, T. B., Ballard, H. L., & Enck, J. W. (2015). Can citizen science enhance public understanding of science? *Public Understanding of Science*, *25*(1), 2–16. https://doi.org/10.1177/0963662515607406

Britch, D., Gechev, I., & Conrey, J. (2023, January 30). *What is .NET MAUI? - .NET MAUI*. Microsoft Learn. https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui?view=net-maui-7.0

Carrion, D., Pessina, E., Biraghi, C. A., & Bratic, G. (2020). Crowdsourcing water quality with the SIMILE app. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, *XLIII-B4-2020*, 245–25. https://doi.org/10.5194/isprs-archives-XLIII-B4-2020-245-2020

Chen, X., Ji, Z., Fan, Y., & Zhan, Y. (2017). Restful API Architecture Based on Laravel Framework. Journal of Physics: Conference Series, 910, 012016. https://doi.org/10.1088/1742-6596/910/1/012016

Cloudmersive (n. d. -a) *Small Business Pricing*. Retrieved March 3, 2023, from https://cloudmersive.com/products

Cloudmersive (n. d. -b) *Small Business Pricing*. Retrieved March 3, 2023, from https://cloudmersive.com/pricing-small-business

eBird. (2021). *2021 year in Review: EBird, Merlin, Macaulay Library, and birds of the world*. eBird. Retrieved November 18, 2022, from https://ebird.org/news/2021-year-in-review#:~:text=To%20those%20116k%20new%20eBirders,forward%20through%20their%20monthly%20contributions

Environment: Embajada de Costa Rica en dc. Environment | Embajada de Costa Rica en DC. (n.d.). Retrieved November 18, 2022, from http://www.costarica-embassy.org/index.php?q=node%2F12
Figma. (n.d.). *Figma: the collaborative interface design tool.* Retrieved January 1, 2023, from https://www.figma.com/

Firebase (n. d.-a) *Firebase*. Retrieved March 3, 2023, from https://firebase.google.com/

Firebase (n. d.-b). *Firebase Pricing*. Retrieved March 3, 2023, from https://firebase.google.com/pricing/

Frost, Rosie. (2020, November 18). "How Did Costa Rica Become the Greenest, Happiest Country in the World?" Euronews. www.euronews.com/green/2020/11/18/how-did-costa-rica-become-the-greenest-happiest-country-in-the-world.

Garrett, J. J. (2010). *The elements of user experience* (2nd ed.). New Riders Publishing.

GitHub. (n.d.-a). *GitHub features: the right tools for the job*. Retrieved March 1, 2023, from https://github.com/features

GitHub. (n.d.-b). *GitHub Issues · Project planning for developers*. Retrieved March 1, 2023, from https://github.com/features/issues

Gong, S. P., & Gibbons, A. S. (2016). An Architectural Experience for Interface Design. *Educational Technology*, *56*(5), 26–34. http://www.jstor.org/stable/44430546

James, M., & Walter, L. (2021). *Scrum Reference Card*. https://scrumreferencecard.com/ScrumReferenceCard.pdf

Microsoft. (2023, February 28). *Visual Studio: IDE and Code Editor for Software Developers and Teams*. Visual Studio. Retrieved March 1, 2023, from https://visualstudio.microsoft.com/

Oberhauser, K., & Caldwell, W. (n.d.). *The Monarch Larva Monitoring Project: Citizen Scientists*

Monitor monarch butterflies. CitizenScience.gov. Retrieved November 18, 2022, from https://www.citizenscience.gov/monitor-monarchs/#

Peterson, J. (2020). *vTree* (Version 3.2.1) [Mobile app]. Apple Store. https://apps.apple.com/us/app/vtree/id576191197

"Costa Rica: The 'Living Eden' Designing a Template for a Cleaner, Carbon-Free World." (2019, September 20). UNEP. www.unep.org/news-and-stories/story/costa-rica-living-eden-designing-template-cleaner-carbon-free-world.

Statcounter. (n.d.). *Mobile operating system market share Costa Rica*. StatCounter Global Stats. https://gs.statcounter.com/os-market-share/mobile/costa-rica

Xcode Overview. (n.d.). Apple Developer. https://developer.apple.com/xcode/

Appendix A

Developer Manual Written in the Markdown Language

```
# [Arboles Maui](https://arbolesmagicos.org)

This is the developer manual for ArbolesMAUI. Its goal is to provide a comprehensive guide through all of the code and tools used, intended to facilitate a smooth

handoff between development teams.


## Purpose

This project aims to port the existing Arboles Magicos iPhone app to the Android platform, while also adding new features and improving the overall codebase. Our goal is to ensure that the app is easy to maintain and expand upon in the future.

#### Development Environment/Tools

![Microsoft Visual Studio](https://img.shields.io/badge/Microsoft%20Visual%20Studio-%20-blue?style=for-the-badge&logo=visual-studio&logoColor=white&color=5C2D91)
![GitHub](https://img.shields.io/badge/Github-%20-blue?style=for-the-badge&logo=github&logoColor=white&color=181717) ![.NET MAUI](https://img.shields.io/badge/.NET%20MAUI-%20-blue?style=for-the-badge&logo=visual-studio&logoColor=white&color=512BD4)
![Git](https://img.shields.io/badge/git-%20-blue?style=for-the-badge&logo=git&logoColor=white&color=F05032)

#### Database Deployment/Management

![Google Firebase](https://img.shields.io/badge/Firebase-%20-blue?style=for-the-badge&logo=firebase&logoColor=white&color=ffca28)

#### Development Languages
```

```
![C#](https://img.shields.io/badge/C%20Sharp-%20-blue?style=for-the-badge&logo=c-sharp&logoColor=white&color=239120)
![XAML](https://img.shields.io/badge/XAML-%20-blue?style=for-the-badge&logo=xaml&logoColor=white&color=0C54C2)
```

Contact

In the event that anything is unclear after reading through this manual, here's a list of our contacts:

```
| Name                                                | Email
|
|-----------------------------------------------|-------------------------
---|
| [Joe Dobbelaar](https://github.com/r2pen2)      | joedobbelaar@gmail.com
|
| [Lex Graziano](https://github.com/lexgraziano) |
alexismgraziano@gmail.com |
| [Cole Parks](https://github.com/cfp02)          | coleparks13@gmail.com
|
| [Jared Chan](https://github.com/CarrotPeeler)   | jared24mc@gmail.com
|
```

Tools

The following is a collection of all tools used in this project as of 2/26/23:

```
- [Microsoft Visual Studio](https://visualstudio.microsoft.com/):
Microsoft's comprehensive integrated development environment (IDE) that
provides a range of tools and services for building applications across
various platforms and languages.

- [.NET
MAUI](https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui?view=net-
```

maui-7.0): Microsoft's new cross-platform framework for building native mobile and desktop apps with C# and XAML. Write once, run anywhere.

- [Google Firebase](https://firebase.google.com/): Google's cloud based platform for developing and distributing applications. The entire ArbolesMAUI back-end is built on Google Firebase.

- [Cloudmersive Image API](https://cloudmersive.com/image-recognition-and-processing-api): API for ranking images on how likely they are to be NSFW.

File System / Project Structure Overview

The .NET Multi-platform App UI (MAUI) project structure is designed to provide a unified development experience across multiple platforms, including iOS, Android, Windows, and macOS. The project structure is built around a single project file that includes platform-specific folders, such as iOS, Android, and macOS, each containing platform-specific implementations of the shared codebase. This approach allows developers to write a single codebase that can be shared across multiple platforms while still enabling platform-specific implementation where necessary. Additionally, the project structure includes a shared project folder, which contains the majority of the codebase that is platform-agnostic, as well as any shared assets, resources, or dependencies.

Application Architecture

In the development of "ArbolesMAUI," the code is divided into two main sections: a back-end or database section responsible for managing the application's data, and a front-end section that handles the user interface and overall user experience. The database section is responsible for storing and retrieving data, managing user authentication and authorization, and ensuring data consistency and integrity. The front-end section, on the other hand, is responsible for presenting the data in a visually appealing and intuitive manner, and handling user interactions. By splitting the code into these two distinct sections, developers can focus on each section's specific requirements and ensure that the application as a whole is performant, reliable, and user-friendly.

General

This section covers topics localization, styling, and the static ViewMediator class, which are crucial to ensuring a consistent and high-quality user experience.

Localization

Authorities: [Jared Chan](#contact)

The code for creating localization was heavily adapted and modified from [VladislavAntonyuk](https://github.com/VladislavAntonyuk/MauiSamples/tree/main/MauiLocalization).

Localization is managed by a singleton class, LocalizationResourceManager.

To change the language, perform the following in C#:

```c#
//Setting culture language to English, United States
LocalizationResourceManager.Instance.Culture = new CultureInfo("en-us");
//CultureInfo must take in a BCP 47 Language Tag
```

OR

```c#
//Setting culture language to Spanish, Costa Rica
LocalizationResourceManager.Instance.Culture = new CultureInfo("es-cr");
//CultureInfo must take in a BCP 47 Language Tag
```

Changing the language creates a call to the C# file, AppResources.cs, which is an autogenerated file by ResGen.exe that manages all AppResources.resx files. AppResources.resx files contain the variable name to reference by in the left column and a corresponding translation in the right column. AppResources.resx is the default translation layer file, which contains English translations. AppResources.es.resx contains Spanish translations.

Using the TranslateExtension.cs file gives a custom markup tag in xaml, making translation binding simple. You just need to state the localization namespace in the content page parameters and use the ```localization:Translate``` markup tag:

```JSX
<ContentPage
  xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:localization="clr-namespace:MauiLocalization.Resources.Localization"
  x:Class="ArbolesMAUI.Views.SamplePage"
>
  <HorizontalStackLayout>
    <Button Text="{localization:Translate TREE-REPORT-PHOTO-SELECT}"/>
  </HorizontalStackLayout>
```

The binding variable used with the markup tag should correspond to a variable name set in the left column of all of the AppResources.resx files.

Styles

Authorities: [Lex Graziano](#contact) and [Joe Dobbelaar](#contact)

.NET MAUI has partial CSS support, but is best styled using XAML. In the Resources/Styles directory there is a Styles.xaml that is used to create styles.

All styles have types associated with them and a series of Property setters. The x:Key acts like a CSS class name.

```JSX
<Style TargetType="Button" x:Key="buttonBase">
  <Setter Property="TextColor" Value="#0A1930" />
  <Setter Property="BackgroundColor" Value="#fefefe" />
  <Setter Property="FontFamily" Value="OpenSansRegular"/>
  <Setter Property="FontAttributes" Value="Bold" />
  <Setter Property="BorderColor" Value="#0A1930" />
  <Setter Property="BorderWidth" Value="1" />
  <Setter Property="FontSize" Value="14"/>
  <Setter Property="CornerRadius" Value="10"/>
  <Setter Property="Padding" Value="10,10"/>
  <Setter Property="Margin" Value="0,5,5,0" />
  <Setter Property="VisualStateManager.VisualStateGroups">
    <VisualStateGroupList>
      <VisualStateGroup x:Name="CommonStates">
        <VisualState x:Name="Normal" />
```

```
        <VisualState x:Name="Disabled">
          <VisualState.Setters>
            <Setter Property="BackgroundColor" Value="#E4E4E4" />
            <Setter Property="BorderColor" Value="#E4E4E4" />
            <Setter Property="TextColor" Value="#8C8C8C" />
          </VisualState.Setters>
        </VisualState>
        </VisualStateGroup>
    </VisualStateGroupList>
  </Setter>
</Style>
```

TO attach the buttonBase style to a button, set the style prop.

```JSX
<Button
  Style="{StaticResource buttonBase}"               // <-- buttonBase is the x:Key on the style defined above
  Text="{localization:Translate FILTER-TREECOLOR}"
  Clicked="OnColorFilterClicked"
/>
```

ViewMediator

Authorities: [Jared Chan](#contact)

The ViewMediator class is static and follows the mediator design pattern. It enables views to talk to each other without coupling them.

The following code demonstrates how references to xaml objects of one view can be stored in the ViewMediator, so other views can access them and change their properties.

```c#
//Culture Page getters/setters

/// <summary>
/// Reference to identify tree species button
/// </summary>
public static Button SelectSpeciesButton { get; set; }

/// <summary>
/// Reference to frame that displays selected color/color of tree species
/// </summary>
public static Frame SpeciesColorBlock { get; set; }
```

Here, xaml object references from the upload page are stored. They can be accessed by the culture page to change their background color or text properties based on which color is selected by the user on the palette.

```c#
private async void OnColorTapped(object sender, EventArgs e)
    {
```

```
        Button btn = (Button)sender;
        TreeColorGroup group = (TreeColorGroup)btn.BindingContext;
        TreeManager firstItem = group.Trees.First();

        //palette.ScrollTo(firstItem, group, ScrollToPosition.Start,
true);

        //separate functionality if culture page is accessed within upload
page
        if (ViewMediator.UploadPageOpen)
        {
            //get the selected color group and set the upload page color
block to that color
            ViewMediator.SpeciesColorBlock.BackgroundColor =
((Button)sender).BackgroundColor;

            //if user identifies by color, return user back to upload
page, prevent tree data from loading
            if (ViewMediator.MethodOfIdentification == "By color")
            {
                ViewMediator.SelectSpeciesButton.Text = "Identify the
tree";
                await Navigation.PopAsync();
            }
        }

        //similar logic to upload page but used for filter page
```

```
        else if (ViewMediator.IsFilterPageOpen)
        {
            if (ViewMediator.MethodOfIdentification == "By color")
            {
                ViewMediator.FilterColorBlock.BackgroundColor =
((Button)sender).BackgroundColor;
                await Navigation.PopAsync();
            }
        }
    }
```

DB

Authorities: [Joe Dobbelaar](#contact)

This section covers everything contained within the DB folder, handling all backend operations of the ArbolesMAUI application. It covers the DBManager class, the ObjectManager abstract class, the DatabaseObjectType enum, and important methods contained within them.

DBManager

Authorities: [Joe Dobbelaar](#contact)

All database interactions are handled in real-time by the DBManager, a factory class with a set of static methods for fetching, storing, and making changes to document data and user authentication. The DBManager also actively listens for changes in the Firestore database and updates local data accordingly.

To store data locally, DBManager has a set of dictionaries associating database document Ids with their respective ObjectManagers that are updated whenever a document is pushed or fetched:

```c#
class DBManager {

    public static Dictionary<string, ColorManager> colorManagers = new Dictionary<string, ColorManager>();

    public static Dictionary<string, ConfigurationManager> configurationManagers = new Dictionary<string, ConfigurationManager>();

    public static Dictionary<string, FloweringCultureManager> floweringCultureManagers = new Dictionary<string, FloweringCultureManager>();

    public static Dictionary<string, GeneralManager> generalManagers = new Dictionary<string, GeneralManager>();

    public static Dictionary<string, MessageManager> messageManagers = new Dictionary<string, MessageManager>();

    public static Dictionary<string, ReportManager> reportManagers = new Dictionary<string, ReportManager>();

    public static Dictionary<string, TreeManager> treeManagers = new Dictionary<string, TreeManager>();

    public static Dictionary<string, UserManager> userManagers = new Dictionary<string, UserManager>();

    public static Dictionary<string, ZoneManager> zoneManagers = new Dictionary<string, ZoneManager>();

}
```

The DBManager also abstracts the creation of ObjectManagers, which are discussed in greater detail below. To create an ObjectManager using the

DBManager's methods, call DBManager.getObjectManager() for the requested object type.

```c#
class DBManager {
    /// <summary>
    /// Create a ReportManager that references a specific Report on the database
    /// </summary>
    /// <param name="id">ID of Report document</param>
    /// <returns>A ReportManager with a docRef to desired Report</returns>
    public static ReportManager getReportManager(string? id, DocumentSnapshot? docSnap = null) {
        if (reportManagers.ContainsKey(id)) {
            return reportManagers[id];
        }
        return new ReportManager(id, docSnap);
    }
}

// Get a ReportManager for the report with documentId "myReport"
ReportManager myReportManager = DBManager.getReportManager("myReport");
GeoPoint myReportLocation = myReportManager.Location; // Will be null until data is fetched
await myReportManager.fetchData();
myReportLocation = myReportManager.Location; // Location from Database
```

...and if you want to create an ObjectManager with existing data, you can pass in that object's Firebase DocumentSnapshot:

```c#
DocumentReference myReportReference =
DBManager.client.Collection("AMReport").Document("myReport");

DocumentSnapshot myReportSnap = await
myReportReference.getDocumentSnapshot();

ReportManager myReportManager = DBManager.getReportManager("myReport",
myReportSnap);

// All fields in myReportManager will me filled with data from the
DocumentSnapshot

GeoPoint myReportLocation = myReportManager.Location; // Location from
Database
```

To listen for changes on the database, the DBManager sets up a FirestoreChangeListener for the entire "AMReport" collection. When the entire "AMReport" collection is fetched by DatabaseObjectType (explored in greater detail below), the listenToReports() method is called.

When changes are detected on the database, the Listener passes a new snapshot of the entire collection to the callback function. Here, the callback function is defined anonymously.

```c#
public static void listenToReports() {

    // Create a FirestoreChangeListener that handles messages from DB for
added or removed reports

    FirestoreChangeListener reportListener =
DatabaseObjectType.Report.getCollectionReference().Listen((snapshot) =>

        // For each change that ocurred..
```

```
        foreach (DocumentChange change in snapshot.Changes) {

            Action changeAction = () => Console.WriteLine("No change")

            if (change.ChangeType == DocumentChange.Type.Added) {

                // There's a new report! Add it to the dictionary and
place a pin on the map

                if
(!DBManager.reportManagers.ContainsKey(change.Document.Id)) {

                    ReportManager rm =
DBManager.getReportManager(change.Document.Id, change.Document);

                    DBManager.reportManagers[change.Document.Id] = rm;

                    changeAction = () => MapUtil.AddNewPin(rm);

                }

            } else if (change.ChangeType == DocumentChange.Type.Modified)
{

                // This report was altered. Change the dictionary entry.

                ReportManager rm =
DBManager.getReportManager(change.Document.Id, change.Document);

                DBManager.reportManagers[change.Document.Id] = rm;

            } else if (change.ChangeType == DocumentChange.Type.Removed) {

                // Report deleted! Remove it from the dictionary and
remove the pin from the map

                DBManager.reportManagers.Remove(change.Document.Id);

                changeAction = () =>
MapUtil.RemovePin(change.Document.Id);



            // Execute update action on main thread (important for
interacting with Map UI)

            MainThread.BeginInvokeOnMainThread(changeAction);
```

```
        }
    });
}
```

ObjectManagers

Authorities: [Joe Dobbelaar](#contact)

ObjectManager is an abstract class that is extended by several subclasses. To get an ObjectManager for a specific document, use the DBManager as described above.

As of 2/28/23, only the following subclasses are used the application:

```c#
public class ColorManager : ObjectManager { ... }

public class ReportManager : ObjectManager { ... }

public class TreeManager : ObjectManager { ... }

public class UserManager : ObjectManager { ... }
```

The constructor for the ObjectManager just takes a DatabaseObjectType. All ObjectManagers contain the following fields and methods:

```c#
public class ObjectManager {

        protected DatabaseObjectType objectType;    // Type of object that this ObjectManager refers to

        public string documentId;                   // Id of document that this ObjectManager points to
```

```
        public DocumentReference docRef;            // Reference to this
ObjectManager's document (if there's a documentId)

        protected DocumentSnapshot docSnap;         // Snapshot of this
document (if fetched)


        public async Task<DocumentSnapshot> getDocumentSnapshot() { }   //
Get the document snapshot by DocumentReference

        public async Task<bool> documentExists() { }                    //
Get whether or not a document with this ID exists on the Database

        public void changeCollection(string newCollection) { }          //
Change the collection that this ObjectManager refers to (only used to move
data from one collection to another)

        public async Task<string> push() { }                            //
Call this.toDictonary() and push data to current collection

}
```

All subclasses must implement the toDictionaryMethod(). This is because the Firestore database is very picky about what kind of data it is given. All data sent to Firestore must be JSON parseable, so we turn the ObjectManager into a Dictionary<string, object> before pushing it to the DB. Since every ObjectManager has different fields, this method must be implemented in each subclass.

```c#
// ObjectManager abstraction

public class ObjectManager {

        public abstract Dictionary<string, object> toDictionary();

}


// Implementation example
```

```
public class UserManager : ObjectManager {

    public override Dictionary<string, object> toDictionary() {

        Dictionary<string, object> data = new Dictionary<string, object>()
{

            { "isAdmin", this.isAdmin },

            { "mail", this.mail },

            { "name", this.name },

            { "createdAt", this.createdAt },

            { "contributions", this.contributions },

            { "isBanned", this.isBanned }

        };

        return data;

    }

}
```

DatabaseObjectType

Authorities: [Joe Dobbelaar](#contact)

DatabaseObjectType is an enum used by ObjectManagers to keep track of and call methods related to their specific type (Users, Reports, etc.). All ObjectManagers have their DatabaseObjectType automatically assigned upon creation.

DatabaseObjectType.js contains the enum as well as a public static class DatabaseObjectTypeExtensions. DatabaseObjectTypeExtensions allows for several methods to be called on DatabaseObjectType as if it were a class. Ex:

```c#

// Database interaction methods

CollectionReference colorCollection =
DatabaseObjectType.Color.getCollectionReference(); // User by all
ColorManagers to get their collection

DatabaseObjectType.Color.fetchAll(); // Async fetches the entire Color
collection, creates colorManagers, and saves all data into DBManager's
colorManagers Dictionary


// Misc methods (unused as of 2/28/23)

string colorString = DatabaseObjectType.Color.toString();   // colorString
= "AMColor"

DatabaseObjectType colorEnum = DatabaseObjectType.getValue("AMColor");
// Inverse of toString()

```


Firestore Database

Authorities: [Joe Dobbelaar](#contact)

ArbolesMAUI uses the same Firestore database as all previous versions of the Ojeadores app. To distinguish between collections used in ArbolesMAUI and older versions, our collections all start with "AM".

| Collection Name | Utility
|

|----------------------|---
------|

| AMColor | Arboles Magicos color catalogue
|

| AMConfiguration | Unused
|

| AMFloweringCulture | Unused
|

| AMGeneral | Unused
|

| AMMessage | Unused
|

| AMReport | Map markers, contain lists of "contributions"
|

| AMTree | Arboles Magicos tree catalogue
|

| AMUser | Data on all users, including their
"contributions" |

| AMZone | Unused
|

Firebase Storage

Authorities: [Joe Dobbelaar](#contact)

All storage operations are handled by—you guessed it—the DBManager! While it was a terrible struggle to get working at first, uploading and deleting photos has been abstracted so much that it should now be very straight forward.

DBManager contains these three static methods for interacting with storage.

```c#
/// <summary>
```

```
/// Upload an image to the reports directory in storage

/// </summary>

public static async Task<string> uploadImage(string fileName, byte[]
byteArray) {

    var task = new FirebaseStorage("ojeadores-6ee96.appspot.com", new
FirebaseStorageOptions { ThrowOnCancel = true })

        .Child("reports")

        .Child(fileName)

        .PutAsync(new MemoryStream(byteArray), CancellationToken.None,
"image/jpg");

    // await the task to wait until upload completes and get the download
url

    string downloadUrl = await task;

    return downloadUrl;

}



/// <summary>

/// Remove an image from the reports directory in storage

/// </summary>

public static async void deleteImage(string fileName) {

    var task = new FirebaseStorage("ojeadores-6ee96.appspot.com", new
FirebaseStorageOptions { ThrowOnCancel = true })

        .Child("reports")

        .Child(fileName).DeleteAsync()

    // await the task to wait until deletion completes

    await task;

}
```

```
/// <summary>

/// Download an image from any directory in storage

/// </summary>

public static async Task<string> downloadImage(string directory, string
fileName)

{

    var task = new FirebaseStorage("ojeadores-6ee96.appspot.com", new
FirebaseStorageOptions { ThrowOnCancel = true })

        .Child(directory)

        .Child(fileName)

        .GetDownloadUrlAsync()

    // await the task to wait until upload completes and get the download
url

    string downloadUrl = await task;

    return downloadUrl;

}
```

They're a little gross looking, but they're really easy to use.

```c#
byte[] imageByteArray; // Any image should be able to be turned into a
byte array

string imageUrl = await DBManager.uploadImage("myImage.jpg",
imageByteArray); // Upload image with name myImage.jpg

string theSameUrl = await DBManager.downloadImage("reports",
"myImage.jpg"); // Get that image's download url (if needed)

await deleteImage("myImage.jpg"); // Delete the image so we're not taking
up any unnecessary storage
```

```

#### Firebase Authentication

Authorities: [Joe Dobbelaar](#contact)

While at first we had hoped to use Google as a sign-in method, we ran into some issues with the .NET MAUI in-app web browser that set off a bunch of red flags on Google's end. Since this didn't work, we decided to just use an email and password sign-in scheme.

All authentication is done through Firebase, so we're not worried about handling any sensitive user information.

DBManager has a GoogleAuthenticationClient that is used by the Login Page. When a user tries to login, the GoogleAuthenticationClient handles everything. If the login is successful, we fetch the document from AMUser associated with the user's UID returned from the login attempt. If that document doesn't exist, we make one. The currentUserManager is stored in the DBManager and a listener is set up to catch any changes. The Signup Page does much the same, but also adds the user to the authentication panel if their account didn't already exist.

### About Arboles Page

Authorities: [Cole Parks](#contact)

The about Árboles Mágicos page is the leftmost page in the appShell navigation. The data (text) from this page was taken from the published *\*Ojeadores\** app, and contains a simple paragraph about the organization, along with the logo and other graphics requested. Under the text are the Árboles Mágicos social icons, which redirect you to the organization's
```

website, Facebook page, Instagram page, and email address in the default email client.

Culture Page

Provides a visually appealing way to explore and view tree species by color. This is the second left most page on the AppShell navigation bar. The design for this page was taken and modified from Diana Zuleta.

Palette

Authorities: [Jared Chan](#contact)

The palette page has two basic elements: a top control bar with a search bar and a button to reload the view, and a collection view.

The collection view loads the colors and their corresponding tree species using MVVM. The collection view is set up to use groups to categorize trees by their color.

The entire culture page binding context is set to an instance of the ```CultureViewModel``` class. The collection view's item source is data bound to a bindable property of the class instance, ```TreeColorGroups```. This property is set at instance creation and is a list of ```TreeColorGroup``` model objects. The model class, ```TreeColorGroup```, contains bindable properties that represent the group color, order, color id, etc., and the class inherits from ```List<TreeManager>```, meaning the class also contains a list of ```TreeManager```. Using this MVVM format for grouping enables the collection view to populate and categorize data by color. For each individual tree that appears under a color tab in the collection view, their data is populated using bindable properties from their corresponding ```TreeManager```.

To clarify, ```TreeColorGroup``` is the binding context for the collection view group headers. ```TreeManager``` is the binding context for each item in the collection view.

Populating and removing items from the collection view is handled via commands sent to the culture page's viewmodel, ```CultureViewModel```.

****Commands currently implemented****

1) ```AddOrRemoveGroupDataCommand```: populates a color header, when tapped on the palette, with ```TreeManager``` items; clears the items of all other color header groups.

2) ```SearchForTreesCommand```: handles which color header groups are populated and which trees appear under each based on the searched tree name.

3) ```ClearItemsCommand```: clears the items of all color group headers; used for refreshing the culture page palette and clearing the search bar entry text.

Tree Detail Page

Authorities: [Lex Graziano](#contact)

The Tree Detail Page only accessible through the [Culture Page](#culture-page). It loads data for the tree selected from [DBManager's](#dbmanager) treeManagers Dictionary.

The top of the page shows a carousel of the focused tree's official Arboles Magicos pictures. In the middle of the page is a description of the tree, along with a calendar of when the tree flowers. This calendar starts off empty and fills in green blocks for each week in the tree's floweringWeeks array.

To add or modify trees, Arboles Magicos should access the [Firestore Database](#firestore-database) and add an AMTree. The Culture page and Tree Detail Page will render all trees in this collection.

Map Page

Authorities: [Cole Parks](#contact), [Jared Chan](#contact), and [Joe Dobbelaar](#contact)

The Map Page is the main screen of ArbolesMAUI. It displays the Microsoft.Maui.Map object stored in ViewMediator to the user, gives them a set of map controls, and the option to upload photos.

Loading the Map

Authorities: [Joe Dobbelaar](#contact)

Due to some issues we had with keeping the map consistent between the Map Page and the Location Picker (both should display the same map, but pins were only getting added on whichever the user was currently looking at), The Map Page places the map stored in [ViewMediator](#viewmediator) in a frame when the page first loads.

We then load all pins from the database.

```JSX
<Frame x:Name="mapFrame" Padding="0" Margin="0"/>
```

```c#
public MapPage()
{
    InitializeComponent();
    mapFrame.Content = ViewMediator.Map; // Set the content of the mapFrame to the Map in the ViewMediator
```

```
    MapUtil.AddAllPinsFromDB();
}
```

MapUtil

Authorities: [Cole Parks](#contact), [Jared Chan](#contact), and [Joe Dobbelaar](#contact)

MapUtil is a static class for interacting with the [ViewMediator's](#viewmediator) Map object. MapUtil is intended to abstract all of the disgusting logic necessary for making changes to the Microsoft.Maui.Map object. It contains methods for adding pins, removing pins, getting all pins from the database, updating the map in real-time when new pins are added by other users, filtering pins, and more.

Adding / Removing Pins

Authorities: [Joe Dobbelaar](#contact)

The addNewPin() and removePin() methods have been abstracted enough that they should be really easy to work with, though they were a nightmare to implement.

To add a pin to the map, call addNewPin() on a reportManager. To remove a pin, call removePin() on a reportId.

```c#
ReportManager myReport = DBManager.getReportManager("report1");
MapUtil.addNewPin(myReport);   // Reads report and adds a pin of the correct color to the map
MapUtil.removePin(myReport.documentId); // Removes the pin associated with report of given ID
```

Please take a look at how addNewPin() works. Adding and removing pins MUST happen on the main thread, otherwise the change won't be reflected on the map's UI. The pin will still be "added" to the map, but you won't be able to see it. This was such an unbelievably frustrating problem to solve. I wouldn't wish trying to figure it out again on my worst enemy.

```c#
/// <summary>
/// Authored by Jared Chan & Joe Dobbelaar
/// Adds a new pin to the map
/// </summary>
public static async void AddNewPin(ReportManager report)
{
    // First we create a new pin from the ReportManager provided
    var pin = new CustomPin() {
        Report = report,
        Type = PinType.Place,
        Location = new Location(report.Location.Latitude,
report.Location.Longitude),
        Label = "",
        Address = "",
        ImageSource = report.getPinImageSource(),
    };

    // We attach listener to the pin that displays a TreePopup when
clicked
    pin.MarkerClicked += async (s, args) =>
    {
```

```
        args.HideInfoWindow = true;
        if (ViewMediator.UploadPageOpen)
        {
            ViewMediator.LastClickedPinInLocationselector = pin;
            ViewMediator.ReportToAddTo = report;
        }
        // TreePopup takes a report for data binding
        TreePopup popup = new TreePopup(report);
        popup.BindingContext = report;
        Application.Current.MainPage.ShowPopup(popup);
    };

    // THIS IS SO IMPORTANT!!! The pin MUST be added on the MAIN THREAD.
    // If we don't use the main thread, the application will recognize
that a pin was added but it will NOT update the UI.
    MainThread.BeginInvokeOnMainThread(() => {
        ViewMediator.Map.Pins.Add(pin);
    });

    // We can also send a push notification if we want
    if(ViewMediator.FirstMapLoad == false)
PushNotifUtil.AddPushNotification(report);
}
```

Camera Button

Authorities: [Joe Dobbelaar](#contact)

The camera button on the map takes the user to the upload page. So long as the user is signed in and isn't banned (a boolean on their AMUser document), the camera button will be shown.

In the event that a user gets banned while the map is open, we still check that a user isn't banned before pushing any contributions to the database.

Map Controls

Authorities: [Cole Parks](#contact)

The map controls are contained in an expander object with a caret as the icon. When the icon is tapped, the three control functions are displayed:

1. Center map on user

    This uses the last known location of the user's phone using the gps, and sets the center of the map to this location.

2. Center map on searched location

    When tapped, a search bar pops up at the top of the map that lets the user search for a Google Maps location (Google Maps Places API). These places are autofilled into a search results VerticalStacklayout, and tapping on one centers the map on that location.

3. Filter

    When tapped, an instance of the Filter Page is opened, which allows the user to select filters and update the visible map pins accordingly. See the Filter Page section for more information.

Filters

Authorities: [Jared Chan](#contact)

The filter page is a separate page that applies filters to the map page. Filters are currently underdeveloped and somewhat broken. The current

implementation for filters is stored in the static utility class, ```MapFilterUtil```.

Three methods currently exist:

1) ```FilterByColor(Color color)```: removes all pins from the map and then re-adds only the pins that match the given color parameter.

2) ```FilterByName(string name)```: removes all pins from the map and then re-adds only the pins that match the given string name parameter.

3) ```ReloadAllMapPins()```: re-adds all existing pins (broken implementation)

Known issues: Filter implementation should not delete or add pins from the map due to potentially bad interaction with the live-updates database listener. Changing implementation to instead hide or show pins should fix the current issues. However, toggling pin visibility is a known challenge to implement.

Location Searching

Authorities: [Jared Chan](#contact)

Location searching functionality utilizes Google Places API. All functions that make requests and handle responses from the API are stored in the static utility class, ```PlacesUtil```.

Google Places API Services Used:

1) AutoComplete: Returns a list of suggested of addresses/places based on a given string address/place (used for when users start searching for a location by address/name)

2) Search.TextSearch: Returns the specific location information, including coordinates, of given address/name of a place (used when the user selects a suggested address/name returned by AutoComplete) - NOTE: AutoComplete

does not give coordinate information in its responses, which is why TextSearch is used to perform a backward search.

How to send requests and handle responses for AutoComplete:

```c#
//send request of data to API
PlacesAutoCompleteRequest request = new PlacesAutoCompleteRequest
{

    Key = API_KEY,
    Input = searchBar.Text,
    Language = lang,
    LocationBias = new LocationBias
    {
        Location = new Coordinate(currLoc.Latitude, currLoc.Longitude),
        Radius = searchRadius
    }
};

//Retrieve response from our request
PlacesAutoCompleteResponse response = await
GooglePlaces.AutoComplete.QueryAsync(request);
```

You can then iterate through the list of predictions to obtain the returned readable data:

```c#
//populate list view search results
foreach (Prediction pred in response.Predictions.ToList())
{
    readableResponseList.Add(pred.Description);
}
```

How to send requests and handle responses for Search.TextSearch:

```c#
PlacesTextSearchRequest request = new PlacesTextSearchRequest
{

    Key = API_KEY,
    Query = LocSearchResults.SelectedItem.ToString(),
    Language = lang,
    Location = new Coordinate(currLoc.Latitude, currLoc.Longitude),
    Radius = searchRadius
};

//Handle response from our request
PlacesTextSearchResponse response = await
GooglePlaces.Search.TextSearch.QueryAsync(request);
Coordinate coords = response.Results.FirstOrDefault().Geometry.Location;
```

Upload Page

Authorities: [Cole Parks](#contact)

The Upload Page is a modal that allows a user to upload a photo of a tree to the database, and it opens when the Camera icon on the Map Page is tapped. From top down, there is a photo placeholder (filled in upon selecting or taking a photo) buttons to take a photo or select one from the user's camera roll, a button to open the LocationPicker page to select a location for the tree, a date selector, a button to open the Palette page to select a color for the tree, and a button to upload the tree to the database as well as a cancel button to close the modal.

In addition, the LocationSelector button and the tree identification button are updated with a location and color when the user selects a location and a color, respectively.

Links

Authorities: [Cole Parks](#contact)

The LocationPicker and Palette pages are opened as modals when their respective buttons are tapped. This is done using the NavigationPage stack. This is the code that opens the LocationPicker page, and a similar function exists for the Palette page.

```c#
private async void LocPickerButton_Clicked(object sender, EventArgs e)
{
    // Set the pin selector to the defaults
    ViewMediator.UsePinAsLocation = false;

    // Sets the ViewMediator LocationPickerButton instance to the pointer of the LocationPicker button
```

```
    ViewMediator.LocationPickerButton = LocationPicker;

    await Navigation.PushAsync(new LocationPicker());

}
```

Image Filtering

Authorities: [Joe Dobbelaar](#contact)

We ensure that images uploaded to the database are inoffensive by making calls to the [Cloudmersive NSFW API](#cloudmersive-image-api). When a user selects an image, we send it to the API and block the user from uploading until the API returns a response. The image is allowed to be sent to the database if it gets a score lower than the threshold. Currently, this threshold is really low. Ideally it won't block any images of trees, but there could be tuning necessary.

The API receives a byte array containing the picture. With our current plan, the image may be no larger than 3.5MB. We use the MAUI IImage interface to compress images before sending them to the API.

```c#
private async void checkImage() {

    checkingImage = true; // Note that we're still checking whether or not this image is NSFW

    Configuration.Default.AddApiKey("Apikey", "7642d63d-55d5-44db-a768-29986d02bce5");

    var apiInstance = new NsfwApi();

    // Compress image

    MemoryStream imageStream = new MemoryStream(imageByteArray);
```

```
    Microsoft.Maui.Graphics.IImage compressedImage =
PlatformImage.FromStream(imageStream);

    compressedImage = compressedImage.Downsize(300);

    byte[] compressedByteArray = compressedImage.AsBytes();


    try {

        // Not safe for work (NSFW) racy content classification

        MemoryStream compressedStream = new
MemoryStream(compressedByteArray);

        NsfwResult result = await
apiInstance.NsfwClassifyAsync(compressedStream);


        // This is our "threshold". Cloudmersive says anything over .8 is
high certainty NSFW.

        // We found that even a score of .5 was probably enough to stop if
from being appropriate for an app for pictures of trees

        // The threshold as of 3/1/23 is VERY conservative. It will
occasionally block pictures of people from upload

        safeImage = (result.Score < 0.1);


        checkingImage = false;

    } catch (System.Exception apiEx) {

        // If we've reached our quota (or some other error), just let the
image through

        Console.WriteLine("Exception when calling NsfwApi.NsfwClassify: "
+ apiEx.Message);

        checkingImage = false;

        safeImage = true;
```

```
    }
}
```

Cloudmersive only allows us 800calls/month^2. See [Scalability](#cloudmersive-image-api). Images will not be blocked from upload when the quota is reached.

Location Picker

Authorities: [Cole Parks](#contact)

The LocationPicker page exists to allow the user to either select a location on the map, or use an existing location. The page is opened as a modal from the Upload Page, and it contains an instance of map with a crosshair in the center as well as a cancel button and a confirmation button.

1. To use a map location with the crosshairs, the user simply centers the crosshairs on the desired location and taps the confirmation button. The Location of the center of the map Map.VisibleRegion.Center is saved in the ViewMediator class, and the LocationPicker page is closed.

2. To use an existing tree's location, the user taps on a pin on the map. In the MapUtil class, when a pin is clicked, the Report that the pin represents is saved in the ViewMediator class.

```c#
ViewMediator.ReportToAddTo = report;
```

Loading the Map

Authorities: [Joe Dobbelaar](#contact)

To ensure consistency, the Location Picker implements the map the same way the Map Page does. See [Map Page: Loading the Map](#loading-the-map)

Crosshair

The crosshairs is a png image that is placed in the center of the screen. When the map is panned, they stay in the same spot and are exactly the Map.VisibleRegion.Center Location. This location is used when the user taps the confirmation button to select the location.

"Use This Tree"

The "Use This Tree" button appears in the pin popup when a user taps on a pin (only when on the LocationPickerPage). When the user taps the button, the Location of the pin is saved in the ViewMediator class, and both the TreePopup and LocationPicker pages are closed. The LocationPicker button on the Upload Page is updated with the location of the pin, which in this case is taken from the Location stored in the report field of the pin that was tapped.

Events Page

There's an empty page for displaying Arboles Magicos events to the user. Arboles should be able to create events in the [Admin Portal](#admin-portal). These events will show up on the map with special markers, as well as send all users a push notification. For more information, see [Events](#requested-features--recommendations).

Account Page

Authorities: [Joe Dobbelaar](#contact)

When the user is signed in, the Account Page shows the user's personal information and button to sign-out. When the user is signed out, the Account Page shows the same Login Form from the Login Page. The Account Page always shows language options.

We had initially planned for the Account Page to show a list of the user's contributions. Contributions are stored on the user, so this would be easy to implement. We just, unfortunately, didn't have time to add this feature.

Login Page

Authorities: [Joe Dobbelaar](#contact)

Login Page allows the user to sign into an existing account or create a new one. We also attempt to sign the user in automatically if we detect that there is a session active on the device.

Sign-In Workflow

See [Firebase Authentication](#firebase-authentication).

Sign-Up Workflow

See [Firebase Authentication](#firebase-authentication).

Auto Sign-In

As of 3/1/23, the auto sign-in method is really scuffed. None of us knew how it should be done so [Joe](#contact) implemented a system that, as far as we know, works (albeit slowly). When the user logs in, we create an empty file in the application's cache directory called loggedIn.txt. Signing out deletes this file. On startup, if the file exists, we attempt to sign the user in with GoogleAuthenticationClient stored credentials.

There's no doubt a better way to persist a user's login session. We just didn't have time to investigate further and this seemed to work for our purposes.

Admin Portal

Authorities: [Joe Dobbelaar](#contact)

There's an administration portal written in React.js in the Admin/arboles-admin directory. The portal is not currently hosted anywhere.

The portal fetches all contributions and allows anyone with an administrator account (same login as ArbolesMAUI) to delete posts, verify posts, and ban users. There's also an area to add an Event Creation form. See [Moderation Levels](#requested-features--recommendations) and [Events](#events-page).

Challenges

Here's a list of some challenges we encountered and their solutions.

- Adding and removing pins on the map must happen on the main thread! See [Adding / Removing Pins](#adding--removing-pins).

- The Map Page and Location Picker were not showing the same map for the longest time. We solved this by making absolutely everything static and loading the [ViewMediator's](#viewmediator) map into both pages, rather than loading the Map Page's map into the [ViewMedaitor](#viewmediator). See [Loading the Map](#loading-the-map).

- Firebase Storage uploading didn't at all. The solution was to stop using [Google's NuGet Package](https://cloud.google.com/storage/docs/reference/libraries#client-libraries-usage-csharp) and switch to the third party [FirebaseStorage.net](https://github.com/step-up-labs/firebase-storage-dotnet).

- Firebase Authentication didn't work either. The solution was, again, to switch from [Google's Authentication NuGet Package](https://cloud.google.com/storage/docs/reference/libraries#client-libraries-usage-csharp) to a third party package: [FirebaseAuthentication.net](https://github.com/step-up-labs/firebase-authentication-dotnet).

Scalability

Authorities: [Joe Dobbelaar](#contact)

Since ArbolesMAUI uses some external APIs / tools, there will be costs associated with scaling up the application. All current quotas are from the Free-Tier versions of these services.

[Google Firebase](https://firebase.google.com/pricing/)

Google Firebase has a free plan (called "Spark") that is enough to support ArbolesMAUI for the time being. For larger scale projects, they offer a "pay as you" plan that charges based on the number of monthly active users an application has.

Firestore Database

Quota	Free Until	Then
GB Stored	1GB	$0.18/GB/Month
Document Writes	20,000/Day	$0.18/100,000
Document Reads	50,000/Day	$0.06/100,000
Document Deletes	20,000/Day	$0.02/100,000

Firebase Storage

Quota	Free Until	Then
GB Stored	5GB	$0.026/GB
DB Downloaded	1GB/Day	$0.12/GB
File Uploads	20,000/Day	$0.05/10,000
File Downloads	50,000/Day	$0.004/10,000

Firebase Authentication

| Monthly Active Users | Price per MAU|

|----------------------|--------------|

| 0 - 49,999 | $0 |

| 50,000 - 99,999 | $0.0055 |

| 100,000 - 999,999 | $0.0046 |

| 1,000,000 - 9,999,999 | $0.0032 |

| 10,000,000+ | $0.0025 |

[Cloudmersive Image API](https://cloudmersive.com/pricing-small-business#:~:text=To%20test%20the%20API%20and,can%20always%20grow%20from%20there.)

Cloudmersive image recognition API is free up to 800 Calls/month^2

| Quotas vs. Price | $0.00/Month | $19.99/Month | $49.99/Month | etc.
|

|------------------|--------------|------------------|------------------|--|

<<<<<<< HEAD

| API Calls | 800/Month^2 | 20,000/Month^2 | 50,000/Month^2 | [See Pricing](https://cloudmersive.com/pricing-small-business#:~:text=To%20test%20the%20API%20and,can%20always%20grow%20from%20there.) |

=======

| API Calls | 800/Month^2 | 20,000/Month^2 | 50,000/Month^2 | [See Pricing](https://cloudmersive.com/pricing-small-business#:~:text=To%20test%20the%20API%20and,can%20always%20grow%20from%20there.) |

>>>>>>> 1bd41e0862aaee7c9f7e60a51d2d1cb5c7c7b197

| API Call Frequency| 1 Call/Second | 2 Calls/Second | 2 Calls/Second | [See Pricing](https://cloudmersive.com/pricing-small-business#:~:text=To%20test%20the%20API%20and,can%20always%20grow%20from%20there.) |

| Max File Size | +-3.5MB | +-1GB | +-4GB | [See Pricing](https://cloudmersive.com/pricing-small-business#:~:text=To%20test%20the%20API%20and,can%20always%20grow%20from%20there.) |

Requested Features / Recommendations

Arboles Magicos expressed interest in many features that we didn't have time to implement. Here's a complete list of requested features and our thoughts on how they may be implemented.

- Show the user's contributions on the account page

    - This should be easy to do. Links to the user's contribution photos are stored on the user's database document. We didn't have time to design and implement this, but it would be quick to build out and make the Account Page much more useful.

- IOS Version

    - .NET MAUI allows for easy cross-platform development. The current map component, however, is android specific. We didn't have time to create an IOS version of the map, but we're confident that the rest of the app will work on Apple devices without needing much attention. Any developer trying to write for IOS will need a computer running MacOS and XCode.

- Hidden Rating System

    - Users should be able to react to photos. Arboles Magicos would like the reactions to be emojis that users can attach to pictures (hearts, flowers, smiles, etc.). These reactions may contribute to how close to the top of the stack the photo shows up. The photo with the highest hidden rating will be displayed first.

    - It has also been suggested that photos are just displayed in order of when they were posted, newer photos first.

    - Some photos should be "starred" by Arboles Magicos. These starred photos will always show up before un-starred photos.

- Hotspots

    - Zooming out on the map creates a big blob of map markers. It would be nice to have pins turn into a heatmap once the user has zoomed out far enough.

    - The best way we came up with to do this is to assign each marker a coordinate on a 2d grid that covers the entire map. Like Minesweeper, one could determine if a pin is close to another by checking the 8 coordinates surrounding it. This way we wouldn't have to check the distance of every pin against each other to determine if there's a hotspot, only some values in a hashmap that will (likely) be null.

- Moderation levels

    - There should be multiple levels of moderators. Right now, anyone who is an admin can ban users, remove photos, verify photos, etc. There should be multiple permission levels.

- Community Moderation

    - Users should be able to notify each other with notes on their posts. For example, a misidentified tree probably doesn't need to be flagged. Someone else could just press a button and the original poster gets a notification that another user thinks their submission may be incorrect.

    - Keeping administration out of these things will save Arboles Magicos time and let people feel like they're a greater part of the community.

- Events

    - Arboles Magicos should be able to create "Events" that send push notifications to users and show up as special markers on the map.

- Calendar Page?

    - It would be nice to see a calendar with events for when each tree flowers. This may be doable with just an embedded Google Calendar. Using a Google Calendar would also let users subscribe to the calendar and display it on their personal calendars.

Known Issues

The following is a list of known issues with ArbolesMAUI, along with our thoughts on how they may be solved.

- Clearing filters doesn't replace pins on the map

    - We didn't have time to investigate this in detail. All of the reports remain in the DBManager's reportManagers Dictionary, so you should just be able to add every value back to the map.

    - We found that map pins don't get added if the addNewPin() method is not [executed on the main thread](#adding--removing-pins) (because we're changing the UI?). The same goes for removing pins, though, and applying filters seems to work fine. The addNewPin() and removePin() methods should also automatically elevate the call to the main thread.

- Auto Sign-In "works"

    - The current method for persisting a user's login session seems to work, but it could probably use some attention. See [Auto Sign-In](#auto-sign-in).

- NSFW Filtering is too strict

    - We didn't have time to tune the Cloudmersive Image Recognition API. Sometimes it says that non-NSFW images are NSFW, blocking upload. This isn't REALLY an issue, but it would be best to resolve this. See [Image Filtering](#image-filtering).

- White trees have pins that look blank

    - Maybe white tree pins get an outline? Maybe the tree icon is shown in black? There are many ways to solve this.

- You may find that the MAUI ListView is bugged. If you use a CollectionView, everything renders properly.

Publishing / Updating the App

Arboles Magicos is expected to handle publishing, maintaining, and updating ArbolesMAUI. Publishing an app to Google Play is free and easy.

Publishing an app to the App App Store requires applying for permission and an Apple developer account. An Apple developer account costs $100/year. We recommend focusing on the Android version at first so that ArbolesMAUI can get some inexpensive user testing before putting the time, energy, and money required to get an app on the Apple App Store.