TECHNICAL UNIVERSITY OF CRETE
SCHOOL OF ELECTRICAL AND
COMPUTER ENGINEERING

Master's Thesis

VISUALIZATION OF NATURE OBSERVATIONS

Koutsouris Themistokles

Examination Committee:
Professor Antonios Deligiannakis, Supervisor
Professor Katerina Mania
Professor George Chalkiadakis

Chania, March 2025

ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ
ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Διπλωματική Εργασία

ΟΠΤΙΚΟΠΟΙΗΣΗ ΠΑΡΑΤΗΡΗΣΕΩΝ ΦΥΣΗΣ

Κουτσουρής Θεμιστοκλής

Εξεταστική Επιτροπή
Καθηγητής Αντώνιος Δεληγιαννάκης, επιβλέπων
Καθηγήτρια Αικατερίνη Μανιά
Καθηγητής Γεώργιος Χαλκιαδάκης

Χανιά, Μάρτιος 2025

Abstract

Natural parks are an essential part of our environment and provide unique opportunities for people to connect with nature. However, visitors to these parks often struggle to navigate the complex terrain and understand the natural surroundings. To enhance visitors' experience and encourage a deeper appreciation of nature, there is a need for interactive and informative tools that can assist visitors in navigating the park and learning about its unique features.

This thesis presents the design and implementation of a web-based platform that addresses these challenges at the Park for the Preservation of Flora and Fauna of the Technical University of Crete. The platform combines real-time location sharing, user-generated content, educational activities, and interactive mapping to create a comprehensive digital companion for park visitors.
Built using modern web technologies including TypeScript, React, Next.js, Node.js and Redis, the platform demonstrates how contemporary software architecture can enhance environmental education and park exploration.

Περίληψη

Τα φυσικά πάρκα αποτελούν αναπόσπαστο μέρος του περιβάλλοντός μας και προσφέρουν μοναδικές ευκαιρίες για τους ανθρώπους να συνδεθούν με τη φύση. Ωστόσο, οι επισκέπτες αυτών των πάρκων συχνά δυσκολεύονται να πλοηγηθούν και να κατανοήσουν το φυσικό περιβάλλον. Για να βελτιωθεί η εμπειρία των επισκεπτών και να ενισχυθεί η εκτίμηση προς τη φύση, υπάρχει ανάγκη για διαδραστικά και ενημερωτικά εργαλεία που μπορούν να βοηθήσουν στην πλοήγηση και την κατανόηση των μοναδικών χαρακτηριστικών του πάρκου.

Η παρούσα διπλωματική εργασία παρουσιάζει τον σχεδιασμό και την υλοποίηση μιας διαδικτυακής πλατφόρμας που αντιμετωπίζει αυτές τις προκλήσεις στο Πάρκο Διάσωσης Χλωρίδας και Πανίδας του Πολυτεχνείου Κρήτης. Η πλατφόρμα συνδυάζει διαμοιρασμό τοποθεσίας σε πραγματικό χρόνο, περιεχόμενο που δημιουργείται από τους χρήστες, εκπαιδευτικές δραστηριότητες και διαδραστική χαρτογράφηση, δημιουργώντας έναν ολοκληρωμένο ψηφιακό οδηγό για τους επισκέπτες του πάρκου.
Χρησιμοποιώντας σύγχρονες τεχνολογίες ιστού, όπως TypeScript, React, Next.js, Node.js και Redis, η πλατφόρμα καταδεικνύει πώς η σύγχρονη αρχιτεκτονική λογισμικού μπορεί να ενισχύσει την περιβαλλοντική εκπαίδευση και την εξερεύνηση του πάρκου.

Acknowledgments

I am grateful to my advisors, Professor Antonios Deligiannakis, Professor Stavros Christodoulakis, and the committee members, Professor Katerina Mania and Professor George Chalkiadakis, for their invaluable guidance.

Dedicated to my wife, whose love, patience, and encouragement made this possible, and to my family for their support throughout this journey.

Themistokles Koutsouris, March 2025

Table of Contents

1. Introduction

This chapter provides an overview of the project, including the background and motivation, problem statement, aims and objectives, scope and limitations, and the thesis structure.

1.1 Background and Motivation

Natural parks and protected areas are important for both environmental conservation and public recreation.
They are an important part of many communities, providing visitors with opportunities to explore and learn about the natural world. They provide opportunities to experience and appreciate nature, while also supporting biodiversity and ecosystem services. In recent years, there has been increasing interest in the use of technology to enhance the visitor experience in natural parks, particularly through the development of web-based applications.

Many visitors may not have a complete understanding of the park's ecosystem, and may not know how to make the most of their visit. Traditional methods of exploring and learning about natural parks, such as paper maps and interpretive signs, can be limiting and may not offer visitors the interactivity and real-time updates they expect in today's digital age.

The primary motivation behind the development of our app is to empower park visitors with a platform that combines the functions of trip planning, user-generated content, educational activities, real-time navigation and location sharing. By providing a seamless and intuitive interface, the application aims to foster a sense of community among park visitors, promote environmental awareness, facilitate responsible exploration of natural spaces, crowdsource scientific research, support park management efforts in conservation, visitor satisfaction, and efficient resource management. The application's potential to provide valuable data on visitor usage patterns, popular points of interest, and environmental observations can inform park management decisions and contribute to the long-term sustainability of the park.

In summary, the background and motivation for this project lie in the opportunities presented by advances in digital technology and the growing interest in user-generated content, real-time location sharing, and environmental education, all of which can be combined to create an interactive web mapping application that enhances the park experience for visitors and supports park management and conservation efforts.

1.2 Problem Statement

The lack of a comprehensive, user-friendly, and interactive platform for visitors at the Park for the Preservation of Flora and Fauna of the Technical University of Crete presents several challenges that this project aims to address:

1. Limited access to real-time, user-generated observations of flora and fauna, making it difficult for park visitors to discover and learn from the experiences of others.
2. Inadequate tools for trip planning and navigation within the park, resulting in missed opportunities to explore points of interest and difficulty in locating specific areas or trails.
3. Insufficient opportunities for engaging in educational activities related to nature and environmental conservation, leading to a lack of awareness and appreciation for the park's ecosystem.
4. Absence of a real-time location sharing and collaboration system, hindering group exploration and posing potential safety concerns for park visitors.
5. A lack of comprehensive data on popular points of interest and environmental observations, which could inform park management decisions and support conservation efforts.

Furthermore, the app must incorporate user-centered design principles, ensuring that it meets the needs and expectations of visitors. It must include a CMS interface for park content management. It must expose a REST API for scientific data mining. Additionally, the app must be designed with adaptability and scalability in mind, so that it can be content-agnostic and easily utilized for other natural parks in the future.

Therefore, the problem statement of this thesis is: How can modern web technologies be leveraged to develop an interactive web mapping application that enhances the visitor experience in natural parks, while incorporating user-centered design principles and scalability/adaptability for future use?

1.3 Aim and Objectives

The specific objectives of this thesis are to develop a platform that facilitates community-building among park visitors by enabling them to share observations of flora and fauna, thus promoting responsible exploration of natural spaces. The project aims to implement a trip planning feature that allows users to plan visits, discover new trails, and locate points of interest within the park, contributing to a more engaging and enjoyable park experience.

The thesis will incorporate educational activities and games focused on nature to provide users with an entertaining and informative way to learn about the park's ecosystem and the importance of environmental conservation. It will integrate an interactive map and navigation system that supports real-time location sharing and collaboration among users, enhancing safety and fostering group exploration within the park.

User-centered design principles will be incorporated throughout the development process, ensuring that the app meets the needs and expectations of visitors. The project will develop a REST API to expose service endpoints for environmental researchers and external applications to gather and process crowdsourced data.

A CMS interface will be developed to allow park management to curate content and gather data regarding visitor usage patterns, popular points of interest, and environmental observations, supporting conservation efforts, visitor satisfaction, and efficient resource management. Finally, the thesis will demonstrate the potential of modern web technologies for enhancing the visitor experience in natural parks and provide insights into best practices for creating similar applications in the future.

In addition to the specific objectives outlined above, this thesis also aims to achieve several broader goals that contribute to the overall success and potential impact of the web application:

This thesis seeks to promote environmental awareness and responsible behavior among park visitors by providing them with easy access to information on local ecosystems, conservation efforts, and best practices for exploring natural spaces. It aims to foster collaboration and knowledge exchange among users by enabling them to share their experiences, observations, and insights, thus creating a dynamic and engaging online community centered around the park.

The project will develop a scalable and adaptable application architecture that can be easily extended to other parks and natural spaces, allowing for the potential replication and customization of the platform in various contexts and locations. It will contribute to the growing body of research on user-generated content, real-time location sharing, and environmental education by presenting a unique case study that combines these elements within a web mapping application.

Furthermore, this thesis will encourage interdisciplinary collaboration by bringing together expertise from various fields, such as computer science, environmental science, education, and park management, to create a holistic and well-rounded application that addresses the diverse needs of users and stakeholders.

By pursuing these broader aims alongside the specific objectives, the thesis aspires to create a comprehensive and impactful web application that not only enhances the park experience for visitors of the Park for the Preservation of Flora and Fauna of the Technical

University of Crete but also serves as a valuable resource and model for similar initiatives in other natural spaces.

1.4 Scope and limitations

This section outlines the scope of the project and highlights the limitations associated with the development and implementation of the thesis.

1.4.1 Scope

The primary focus of the application is to enhance the park experience for visitors at the Park for the Preservation of Flora and Fauna of the Technical University of Crete. However, the application's architecture and design principles will be adaptable and scalable, allowing for potential extension to other parks and natural spaces.

The application will include features such as observation sharing, trip planning, educational activities, and real-time interactive maps, addressing the specific challenges outlined in the problem statement.

The application is written in Typescript, utilizing React and Next.js as the frameworks of the client app, Strapi.js for the CMS and REST API, and Redis for the backend real time user location service. The study aims to analyze the effectiveness of the application in enhancing the visitor experience in the park.

Usability testing and user feedback will be conducted to ensure that the application meets the needs of various user groups, including casual visitors, nature enthusiasts, educators, and researchers.

The project will support park management in conservation efforts, visitor satisfaction, and resource management by providing valuable data on visitor usage patterns, popular points of interest, and environmental observations.

1.4.2 Limitations

The accuracy and reliability of user-generated content, such as observations and trip plans, may vary, and the application relies on the active participation of users to maintain updated and relevant information.

The application's performance and user experience may be affected by factors such as internet connectivity, device compatibility, and user familiarity with web mapping applications.

The application will primarily focus on the Park for the Preservation of Flora and Fauna of the Technical University of Crete, and some features or content may not be directly transferable or applicable to other parks or natural spaces without adaptation or customization. The potential scalability and adaptability of the application to other parks and

natural spaces may be limited by factors such as variations in park infrastructure, local regulations, and the availability of geographic and environmental data.

The project's timeline and available resources may limit the extent of testing, user feedback, and refinement, potentially affecting the application's overall quality and functionality.

Legal, ethical, and privacy concerns related to user-generated content and real-time location sharing may pose challenges in the development and implementation of the application. It will be necessary to establish appropriate guidelines, user agreements, and data handling practices to address these concerns and ensure compliance with relevant regulations.

The application's effectiveness in promoting environmental awareness and responsible behavior among park visitors is dependent on user engagement and participation in the educational activities, and measuring the long-term impact of these activities on user behavior may be challenging.

By acknowledging the scope and limitations of the project, this thesis aims to develop a comprehensive and impactful web application that addresses the specific needs and challenges of park visitors at the Park for the Preservation of Flora and Fauna of the Technical University of Crete, while also considering the potential for extension and adaptation to other natural spaces.

1.5 Thesis Structure

This thesis is organized into several chapters, each focusing on a specific aspect of the project. The structure of the thesis is as follows:

1. **Introduction**: This chapter provides an overview of the project, including the background and motivation, problem statement, aims and objectives, scope and limitations, and the thesis structure.

2. **Literature Review**: In this chapter, relevant literature and research related to web applications in environmental education, user-generated content in nature observations, real-time location sharing and collaboration, and environmental education and gamification in natural spaces are reviewed and discussed.

3. **Methodology**: This chapter describes the project's methodology, covering aspects such as research design, stakeholders, personas, use cases, and prototypes.

4. **Implementation and Application Features**: Detailed analysis of the app implementation, including frontend and backend development, integration, deployment processes, challenges, and solutions. Presentation of the application

features, including observations sharing, trip planning, educational activities, interactive map and navigation, and real-time location sharing and team collaboration.

5. **Testing and Evaluation**: This chapter aims to systematically analyze how well the application meets its intended objectives, adheres to the specified requirements, and provides a satisfactory user experience.

6. **Conclusion and Future Work**: This chapter summarizes the key findings and contributions of the project and discusses potential future work and enhancements to the application.

7. **References**: This section contains a comprehensive list of all the cited literature and research used throughout the thesis.

8. **REST web services**: This appendix provides the REST API documentation.

2. Literature Review

In this chapter, we review and discuss relevant literature and research related to web mapping applications in environmental education, user-generated content in nature observations, real-time location sharing and collaboration, environmental education and gamification in natural spaces.

2.1 Web Mapping Applications in Environmental Education

Web mapping applications offer a powerful platform for sharing and analyzing environmental data, which can be leveraged to facilitate communication, collaboration, and informed decision-making among diverse stakeholders (Goodchild, M. F. 2007). By integrating spatial data and providing interactive tools for visualization and analysis, these applications can help to bridge the gap between expert knowledge and public understanding of complex environmental issues.

Web mapping applications have proven to be valuable tools in achieving environmental education goals, as they allow users to explore and analyze environmental data in a spatial context, thereby offering a more holistic understanding of the issues at hand. One notable example of the use of web mapping applications in environmental education is the work of Bodzin, Anastasio, and Kulo (2014), who designed Google Earth activities for teaching about the earth and the environment. They found that the use of geospatial technologies, including web mapping applications, can enhance students' understanding of complex environmental concepts and promote spatial thinking skills.

Brown and Weber (2012) explored the use of Public Participation Geographic Information Systems (PPGIS) in national park planning. Their research demonstrated that web mapping applications can be an effective tool for engaging stakeholders, facilitating collaboration, and informing decision-making processes in environmental management.

The potential of web mapping applications for citizen science and Volunteered Geographic Information (VGI) has been widely recognized (Haklay, 2013). For instance, Sullivan et al. (2014) described the development and application of eBird, a citizen science project that uses web mapping tools for collecting, visualizing, and sharing bird observation data. This project has successfully engaged a large community of users, contributing valuable data for bird conservation and research.

Kelling et al. (2015) examined the role of web mapping tools in the eBird project, which gathers bird observation data from volunteers worldwide. The authors found that the geospatial data generated by eBird participants provided valuable information for understanding bird distribution patterns and assessing the effects of environmental changes on bird populations.

In conclusion, web mapping applications have demonstrated their potential as powerful tools for enhancing environmental education, fostering collaboration, and promoting informed decision-making. By leveraging the unique capabilities of these applications, educators and practitioners can better equip learners with the knowledge, skills, and attitudes necessary to navigate and address the complex environmental challenges of our time.

2.2 User-generated Content in Nature Observations

The advent of the digital age has witnessed a paradigm shift in the creation and dissemination of information. Where once information was produced and shared by a handful of gatekeepers, we now live in an era where users are active contributors, generating a plethora of content across various digital platforms. This phenomenon, known as User-Generated Content (UGC), has found its application in a wide array of fields, including environmental science and nature observation.

User-generated content in nature observations often comes in the form of volunteered geographic information (VGI), a type of user-generated content where individuals voluntarily contribute spatial information (Elwood, Goodchild, & Sui, 2012). This is particularly evident in the context of nature observations, where amateur naturalists and citizen scientists contribute valuable data about biodiversity, species distribution, and other ecological phenomena.

The eBird project, as mentioned in the previous section, is a leading example of how UGC and VGI are leveraged in nature observations. Initiated by the Cornell Lab of Ornithology and the National Audubon Society, this global citizen science project allows birdwatchers to input their observations into a centralized database, contributing to a wide range of ecological research and conservation efforts (Sullivan et al., 2014).

The power of UGC extends beyond data collection to fostering a sense of community among users. Platforms like iNaturalist provide a social network for naturalists, where users can share their observations, get help with species identification, and participate in biodiversity conservation projects. By integrating elements of social media, these platforms are not only engaging users but also democratizing science and fostering a sense of environmental stewardship (Silvertown, 2009).

However, despite its potential, UGC in nature observations also poses several challenges. The quality and reliability of user-generated data are among the major concerns, given that contributors vary widely in their level of expertise (Haklay, 2013). Issues related to data privacy, intellectual property rights, and the digital divide are other critical considerations in the use of UGC in nature observations.

In conclusion, user-generated content in nature observations holds tremendous potential for promoting citizen science, enhancing environmental education, and fostering a sense of community among users. By leveraging the power of UGC and addressing its associated challenges, we can harness the collective intelligence of the public to better understand and conserve our natural environment.

2.3 Real-time Location Sharing and Collaboration

Real-time location sharing has grown significantly in popularity with the rise of mobile and web applications. Its applications span various domains, including social networking, logistics, emergency response, and particularly, outdoor activities and nature observation.

In the context of outdoor recreational activities, real-time location sharing can enhance safety, foster a sense of community, and facilitate coordination for group activities. This feature can be particularly beneficial in expansive and potentially hazardous natural parks, allowing participants to track each other's locations, coordinate group activities such as guided tours or educational games, and share a common experience.

When combined with user-generated content, real-time location sharing can provide context-specific information that enhances the experiences of park visitors. Users can share their observations of wildlife, flora, or notable landmarks, along with their location. This user-generated data contributes to a shared knowledge base and invites others to explore the same features or phenomena.

Incorporating these capabilities into our web mapping application aligns with the broader trend of spatially enabled social networking, which leverages geospatial technologies to foster interaction, collaboration, and community-building. With these features, our application serves not just as a navigational tool, but also as a platform for engaging with the natural park in a collaborative and interactive manner.

2.4 Environmental Education & Gamification in Natural Spaces

Gamification, the use of game design elements in non-gaming contexts, has been increasingly explored for its potential in enhancing engagement and learning outcomes in environmental education (Deterding, Dixon, Khaled, & Nacke, 2011). In the context of environmental education in natural spaces, games can potentially foster a sense of environmental responsibility and stewardship among players.

Geocaching, an outdoor recreational activity that involves using GPS coordinates to find hidden caches, is an example of how game-like activities can be integrated into environmental learning experiences. This activity can increase interest and engagement in outdoor explorations, and can potentially be used to promote understanding and conservation of natural spaces (O'Hara, 2008).

However, it's critical to acknowledge the challenges associated with gamification. Games must be well-designed, with a balance of challenge and skill, meaningful choices, and clear alignment between game objectives and learning outcomes (Deterding et al., 2011). Further, the effective integration of games into educational practice requires thoughtful planning and guidance from educators to facilitate reflection and deeper learning.

Importantly, research findings on the efficacy of gamification in educational contexts are mixed. For instance, a study by Hanus & Fox (2015) found that while gamification can increase motivation and satisfaction, it may not always lead to improved learning outcomes. This underscores the need for careful and critical implementation of gamification strategies in educational contexts.

In conclusion, while gamification presents exciting opportunities for environmental education in natural spaces, further research is needed to understand how to maximize its benefits, navigate potential challenges, and avoid potential pitfalls.

2.5 Best Practices for Designing & Developing Web-Based Apps in Natural Parks

Based on previous studies and experiences with web-based applications in natural parks, several best practices have been identified for designing and developing effective applications. These include:

Understand the needs and preferences of visitors: To design an effective web-based application, it is important to understand the needs, preferences, and behavior of park visitors. This can be achieved through user surveys, focus groups, and other forms of user research.

Use a mobile-first design approach: Many visitors will access the web-based application on their mobile devices, so it is important to use a mobile-first design approach to ensure that the application is optimized for smaller screens.

Provide clear and concise information: The application should provide clear and concise information about park features, activities, and rules, as well as real-time updates on park conditions.

Use interactive and engaging content: To enhance visitor engagement and learning, the application should include interactive and engaging content, such as videos, images, and educational games.

Enable data collection and sharing: To support scientific research and conservation efforts, the application should enable visitors to collect and share data on park features and wildlife.

By following these best practices, web-based applications in natural parks can provide a more engaging and informative experience for visitors, while also supporting scientific research and conservation efforts. The next chapter describes how these best practices were incorporated into the design and development of the web-based application for the Park for the Preservation of Flora and Fauna of the Technical University of Crete.

3. Methodology

This chapter describes the project's methodology, covering aspects such as research design, stakeholders, personas, use cases, and prototypes.

3.1 Research design

The research design for this thesis follows a user-centered design approach, revolving around the development, testing, and refinement of a web-based application to enhance visitors' experiences in natural parks in general, and of the Park for the Preservation of Flora and Fauna of the Technical University of Crete in particular.

This process primarily entailed the use of paper prototypes, which were thoroughly tested with potential users to gather valuable feedback and thereby improve the application design (Snyder, 2003). Despite the limitations of paper prototypes, primarily their inability to fully emulate the functionality and behavior of the final product, they serve as an instrumental tool during the early design stages. Their inherent capacity for rapid iteration and refinement expedited the design process, and ensured a more effective transition into the development stage.

The paper prototypes were constructed based on personas and use cases identified through stakeholder interviews and user research, an established technique to effectively anticipate user needs and behaviours (Cooper, Reimann, & Cronin, 2007). Personas in this context are fictional characters that represent the target users of the web application, while use cases depict various scenarios that outline the potential interactions between users and the application.

Stakeholder feedback was central to the design process, optimizing the web application's features and functionality to better serve the end users. The feedback was integrated into the design process to enhance the application's potential for improving the visitor experience (Sanders & Stappers, 2008).

The prototypes developed, encompassing representations of the application's interface, functionality, and features, were evaluated via usability testing sessions and feedback surveys. These methods allowed for effective feedback collection and subsequent refinement of the application design

3.2 Stakeholders

The development and implementation of the web-based application designed to enhance natural park visitor experiences involve a diverse range of stakeholders. These stakeholders each have unique interests and perspectives, and their input is invaluable for refining the application and ensuring it meets the needs of all users:

End Users: These are the visitors to the natural parks who will be the primary users of the web-based application. Their feedback on the application's usability, functionality, and relevance to their needs is crucial for the iterative design process.

Park Management: This group includes park rangers, administrators, and other staff members responsible for managing the park and ensuring its sustainability. Their insights into the operational aspects of the park, visitor trends, and potential environmental impacts can greatly inform the design and functionality of the application.

Educators: Teachers, professors, and other educators who may use the application as a tool for environmental education can provide valuable input on features that support learning and engagement.

Environmental Scientists and Researchers: This group can contribute unique insights into the application's potential for data collection and research, including citizen science initiatives.

Local Communities: People living in and around the park may have valuable local knowledge and a vested interest in the park's sustainability. Their input can be useful in shaping the application's approach to community involvement and local impact.

Tourism Authorities: As entities often interested in promoting sustainable tourism, their perspectives can help ensure that the application aligns with broader regional or national tourism strategies.

Tech Developers: Those responsible for the technical development of the application have a vested interest in understanding the needs and desires of other stakeholders to ensure a functional and user-friendly product. They work in the intersection of user needs, technical feasibility, and business requirements.

Involving all of these stakeholders in the design process ensures a more comprehensive understanding of the various needs and preferences associated with the application. Moreover, it allows for a more inclusive and democratic approach to decision-making, encouraging collaboration and a sense of shared ownership over the application.

3.3 Personas

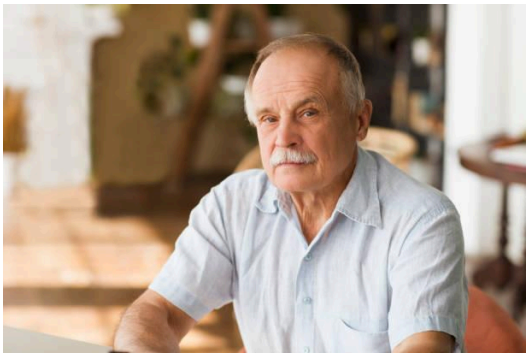
In the design and development of user-centric applications, understanding the users' needs, behaviors, and goals is paramount. One effective approach to encapsulate these aspects is through the creation of 'personas'. A persona, in the context of user-centered design, is a fictional character that represents a user type that might use the site, brand, or product in a similar way. These personas help to align strategy and goals to specific user groups, providing us with a clear understanding of the user's expectations and how they are likely to interact with the system.

In this section, we present a series of personas developed to represent the different types of users anticipated for the web application being developed for the natural park. These personas were created based on research and observation, along with input from various stakeholders, and each persona represents a significant portion of people in the real world. These personas will help us better understand our users' needs, experiences, behaviors, and goals.

Name	Johan
Age	45
Occupation	Park Manager
Interests	Conservation, Public Service, Wildlife Protection
Goals	Efficient park management, Conservation of local ecosystems, Improve visitor experience
Challenges	Budget restrictions, Balancing visitor needs with conservation efforts
Tech Knowledge	High - Uses various technologies for park management
App Interaction	Interested in using the app for visitor data analytics and sharing park updates
Quote	"I hope this app can give me better insights about visitor behaviors and their needs."

Name	Carlos
Age	30
Occupation	Park Guide
Interests	Wildlife, Outdoor Adventures, Education
Goals	Provide informative and engaging tours, Promote responsible behaviors in the park
Challenges	Adapting tours to different visitor groups, Handling unexpected situations
Tech Knowledge	Moderate - Uses technology as part of job, mostly for communication
App Interaction	Can use the app to share tour information and updates, edit metadata through CMS
Quote	"A good tour is informative and fun. The app could help me deliver that."

Name	Dr. Liu
Age	38
Occupation	Environmental Researcher
Interests	Biodiversity, Climate change, Ecology
Goals	Conduct research in the park, Publish findings in academic journals
Challenges	Accessing accurate and comprehensive data, Securing research funding
Tech Knowledge	High - Utilizes advanced technology in research
App Interaction	Interested in accessing detailed environmental data from the REST interface
Quote	"Reliable and accessible data is crucial for my research. I hope this app can provide that."



Name	George
Age	70
Occupation	Retired Engineer
Interests	Family activities, History, Gardening
Goals	Safe and fun outings with grandchildren, Sharing historical and natural knowledge
Challenges	Keeping up with energetic grandchildren, Making sure the activities are suitable for their age
Tech Knowledge	Basic - Uses smartphone mainly for communication and basic searches
App Interaction	Interested in using the app for safety alerts, kid-friendly trails, and learning materials that he can share with grandchildren
Quote	"I'd like an app that can help me share the beauty and history of the park with my grandchildren, safely and enjoyably."

Name	Abby
Age	28
Occupation	Software Engineer
Interests	Hiking, Wildlife, Photography
Goals	Explore new trails, Take good photographs, Learn about the park's wildlife
Challenges	Limited free time, Sometimes gets lost in unfamiliar trails
Tech Knowledge	High - Comfortable with smartphones and applications
App Interaction	Interested in using the mobile web app for navigation and learning about wildlife
Quote	"I want an app that helps me explore new trails and spot wildlife."

Name	Ellen
Age	26
Occupation	Environmental Educator
Interests	Teaching, Nature, Community outreach
Goals	Educate visitors about the importance of conservation, Design engaging learning activities
Challenges	Capturing the attention of diverse learner groups, Making complex concepts understandable
Tech Knowledge	Moderate - Comfortable using technology for educational purposes
App Interaction	Interested in accessing and contributing to educational content in the app
Quote	"I want to leverage the app to make learning about nature fun and interactive."

Name	Maria
Age	35
Occupation	High School Teacher
Interests	Bird watching, Plant identification, Education
Goals	Observe new bird species, Learn and teach about local flora and fauna
Challenges	Needs up-to-date information on wildlife sightings, Difficulty identifying certain plant species
Tech Knowledge	Moderate - Knows basic smartphone operation
App Interaction	Interested in sharing sightings and accessing educational content
Quote	"I need a tool that updates me about bird sightings and helps me identify plants."

3.4 Use Cases

The identified use cases of the system, described using Alistair Cockurn's methodology

Use case 1	Log in
Actor	Visitor
Scope & Level	System, sub-function
Goal in context	The user wants to log in to use the full system's functionality
Precondition	The user has created an account
Basic Flow	1. The user clicks the login button on the top right of the page. 2. The user enters a username and a password. 3. The system validates that the submitted credentials are valid. 4. The system enables the login button. 5. The user clicks the login button. 6. The system checks that the submitted credentials are correct. 7. The system displays the user's account page.

Use case 2	Register
Actor	Visitor
Scope & Level	System, sub-function
Goal in context	The user wants to register to use the full system's functionality
Precondition	The user has not created an account yet
Basic Flow	1. The user clicks the login button on the top right of the page. 2. The user enters an email, a username and a password. 3. The system validates that the submitted credentials are valid. 4. The system enables the register button. 5. The user clicks the register button. 6. The system checks for conflicts with existing user names. 7. The system displays the user's account page.

Use case 3	Find area
Actor	Visitor
Scope & Level	System, sub-function
Goal in context	The user wants to view information on a specific area of the map
Precondition	The user has opened a page with a map containing the area he is interested in
Basic Flow	1. The user clicks/taps on an area. 2. The system displays a popup with high level information on the area. 3. The user clicks/taps on the popup 4. The system opens the specified area page for more information.

Use case 4	Find observation
Actor	Visitor
Scope & Level	System, sub-function
Goal in context	The user wants to view information on a specific observation
Precondition	The system has observations.
Basic Flow	1. The user navigates to the main page. 2. Depending on the observation type, the user clicks/taps the “plants” or “animals” button 3. The system opens the appropriate search page. 4. The user may write in the query field relevant text. 5. The system updates the results in real time. 6. The user may narrow down his results by clicking/tapping on a filter & making a selection. 7. The system updates the results. 8. The user may click on the “created” filter to display only the observations he created. 9. The system shows only observations that the user created. 10. When the user finds the observation of interest, he clicks/taps on its card. 11. The system opens the observation’s data page.

Use case 5	Create observation
Actor	Visitor
Scope & Level	System, sub-function
Goal in context	The user wants to create a new observation
Precondition	The user has an account and is logged in.
Basic Flow	1. The user navigates to the main page. 2. The user clicks/taps on the “new observation” button. 3. The system opens the observation creation page. 4. The user chooses a taxon and writes title, description, and other metadata in the relevant fields. 5. The user may take a picture by clicking on the camera field. 6. The system loads the user’s device camera UI and awaits for image data. 7. The user may update the observation’s position by clicking on the map. 8. The user clicks/taps the “save” button 9. The system uploads the new observation. 10. The system displays the created observation’s page

Use case 6	Create trip
Actor	Visitor
Scope & Level	System, sub-function
Goal in context	The user wants to create a new trip
Precondition	The user has an account and is logged in.
Basic Flow	1. The user navigates to the main page. 2. The user clicks/taps on the “new trip” button. 3. The system opens the trip creation page. 4. The user writes a title, a description, and ticks any relevant observations he created. 5. The user may take a picture by clicking on the camera field. 6. The system loads the user’s device camera UI and awaits for image data. 7. The user creates a trip path by clicking to create new points on the map 8. The system adds points to the path with each click. 9. The user may click on points of the path to remove them. 10. The system removes clicked points from the path. 11. The user clicks/taps the “save” button

	12. The system uploads the new trip. 13. The system displays the created trip's page

Use case 7	Update observation
Actor	Visitor
Scope & Level	System, sub-function
Goal in context	The user wants to change one of his created observations
Precondition	The user has an account, is logged in, and has created at least one observation.
Basic Flow	1. The user finds the observation (use case 4) 2. The user clicks/taps one of his created observations 3. The system opens the observation page 4. The user clicks/taps the "edit" button on the top left of the data section. 5. The system opens the observation edit page. 6. The user updates any data he wishes by clicking and editing the appropriate fields. 7. The user clicks/taps the "save" button. 8. The system updates the observation 9. Alternatively, the user clicks "reset" to cancel any changes. 10. In both cases, the system opens the observation's view page.

Use case 8	Find activity
Actor	Visitor
Scope & Level	System, sub-function
Goal in context	The user wants to find an activity
Precondition	The system has activities.
Basic Flow	1. The user navigates to the main page. 2. The user clicks/taps the "activities" button 3. The system opens the activity search page. 4. The user may write in the query field relevant text. 5. The system updates the results in real time. 6. The user may narrow down his results by clicking/tapping on a filter & making a selection. 7. The system updates the results. 8. When the user finds the activity of interest, he clicks/taps on its card.

	9. The system opens the activity's data page.

Use case 9	Like observation
Actor	Visitor
Scope & Level	System, sub-function
Goal in context	The user wants to like an observation
Precondition	The user is registered and logged in
Basic Flow	1. The user finds the data page of the observation (use case 4) he wants to like. 2. The user clicks the heart icon on the top right of the data section. 3. The system includes the observation in the user's likes, fills the icon and updates liked user count.

Use case 10	Create/Join group
Actor	Visitor
Scope & Level	System, sub-function
Goal in context	The user wants to create a user group or join an existing one
Precondition	The user is registered and logged in
Basic Flow	1. The user clicks on the account button on the upper right corner. 2. The system opens the account page. 3. The user writes a new or existing group code in the "group code" field 4. The user clicks "join". 5. The system creates the group if it doesn't exist. 6. The system adds the user to the group.

Use case 11	Create group alert
Actor	Visitor
Scope & Level	System, sub-function
Goal in context	The user wants to create a distance alert for his group
Precondition	The user is registered and logged in. The user has created or joined a group
Basic Flow	1. The user clicks on the account button on the upper right corner. 2. The system opens the account page. 3. The user types a distance in meters in the “group alert distance” field 4. The user clicks “save” 5. The system will displays the alert radius on the map 6. The system will alert the user if people in the group stray too far away from him.

Use case 12	Find trip
Actor	Visitor
Scope & Level	System, sub-function
Goal in context	The user wants to find a trip
Precondition	The system has trips
Basic Flow	1. The user navigates to the main page 2. The user clicks/taps the “trips” button 3. The system opens the trip search page 4. The user enters in the query field relevant text 5. The system updates the results in real time 6. The user may further narrow down his results by clicking/tapping on a filter and making a selection 7. The system updates the results 8. The user clicks/taps on a result. 9. The system opens the trip’s data page.

Use case 13	Add trip commentary
Actor	Visitor
Scope & Level	System, sub-function
Goal in context	The user wants to add commentary to a trip
Precondition	The user is registered and logged in. The trip exists.
Basic Flow	1. The user navigates to the relevant trip 2. The user scrolls to reach the commentary section 3. The user chooses “you” in the “commentary by” dropdown menu 4. The user enters his commentary in the text box below and clicks “add to your coordinates” 5. The system updates your commentary to include your newly added text 6. The system shows its coordinates on the map

Use case 14	View trip commentary
Actor	Visitor
Scope & Level	System, sub-function
Goal in context	The user wants to view commentary of a trip
Precondition	The trip exists.
Basic Flow	1. The user navigates to the relevant trip 2. The user scrolls to reach the commentary section 3. The user chooses a user (including the Author or himself) from the “commentary by” dropdown menu 4. The system updates the commentary to be of the chosen user. 5. The system shows each commentary’s coordinates on the map

Use case 15	Log in the CMS
Actor	Administrator
Scope & Level	System, sub-function
Goal in context	The user wants to log in the CMS
Precondition	The user has obtained login credentials.
Basic Flow	1. The user opens the CMS page. 2. The system loads the login prompt. 3. The user enters his credentials in the appropriate fields. 4. The system checks that the credentials are correct. 5. The system loads the CMS main page.

Use case 16	Update taxon in the CMS
Actor	Administrator
Scope & Level	System, sub-function
Goal in context	The user wants to update a taxon through the CMS UI
Precondition	The user is logged in the CMS. The taxon exists.
Basic Flow	1. The user chooses “taxon” from the “collection types” list in the CMS UI. 2. The user searches for the desired taxon by clicking the search button on the top left and entering its label 3. The user can narrow down the results by selecting a filter from the filter button on the top left. 4. The system updates the results in real time. 5. The user clicks on a taxon. 6. The system opens the taxon CMS metadata page. 7. The user updates a field. 8. The user clicks the “save” button on the upper right 9. The system updates the taxon.

Use case 17	Get observation list through REST
Actor	Expert
Scope & Level	System, sub-function
Goal in context	The user wants to get an observation list through the REST interface
Precondition	The user has an API token (requested from the CMS administrator) and/or user credentials
Basic Flow	1. The user chooses the appropriate REST request from the documentation. 2. The user enters the REST request in an appropriate console (e.g. Postman), including as parameters the API token or the user credentials, and sends it to the correct REST endpoint of the system. 3. The system checks if the credentials and the request are correct. 4. The system responds with the requested data. 5. The user receives the response in json format.

3.5 Prototyping

At the initial stages of the design process, we drew prototypes of the UI and created storyboards to visualize the user flow.
A storyboard is a representation of a particular interaction sequence. Storyboards enable the better visualization of the interaction and the navigation between screens, while presenting most of the functionality of the system. Moreover, they enable brainstorming and allow changes to occur on the fly if a problem is found. In our user interface design process, the storyboards were used in multiple heuristic evaluations with users broadly conforming to our personas.
We were able to receive feedback from these evaluations and refine our design, making it more user friendly and efficient.

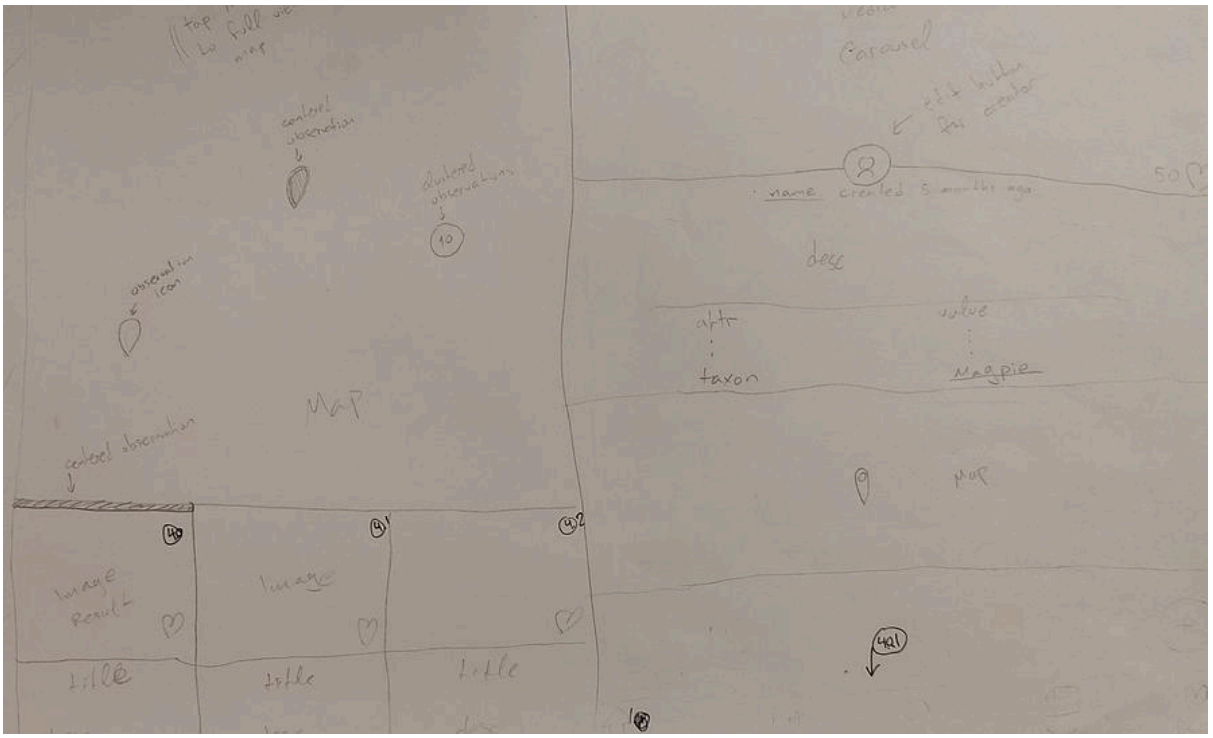


Figure 3.1: Early storyboard sketch, main page & profile page, with numbered branching UI actions

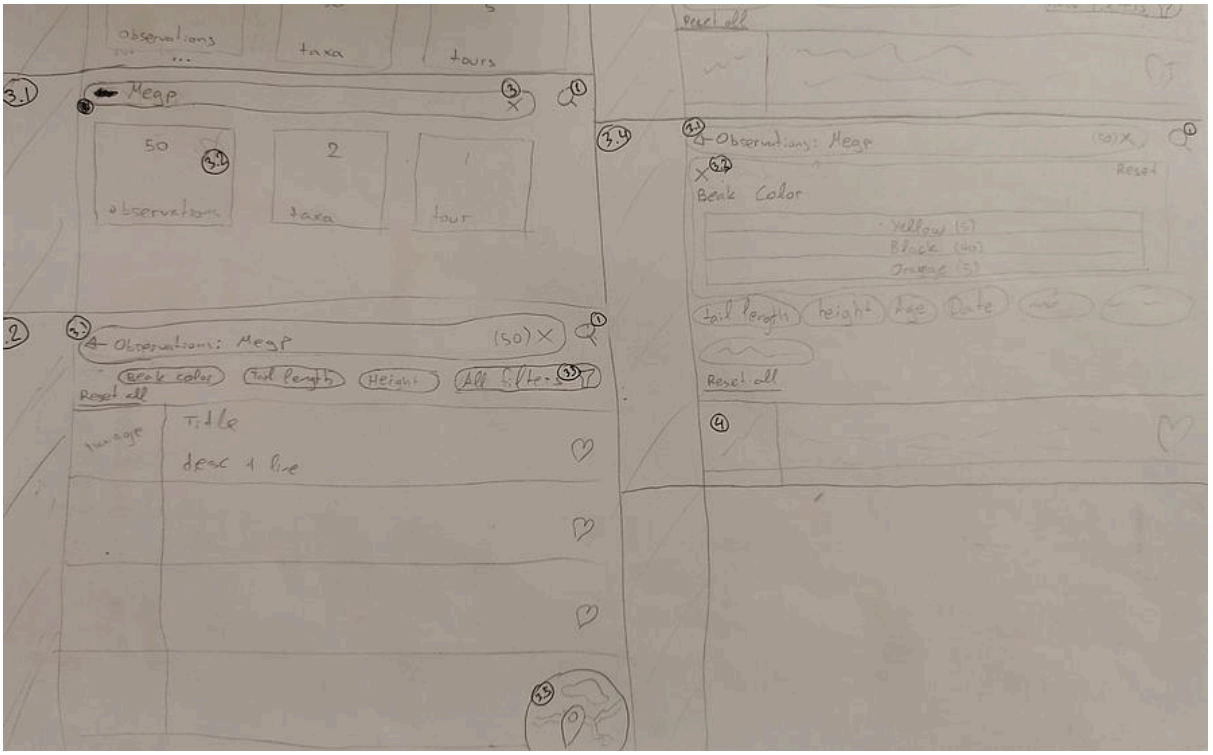


Figure 3.2: Early storyboard sketch, multiple search page states, numbered branching UI actions

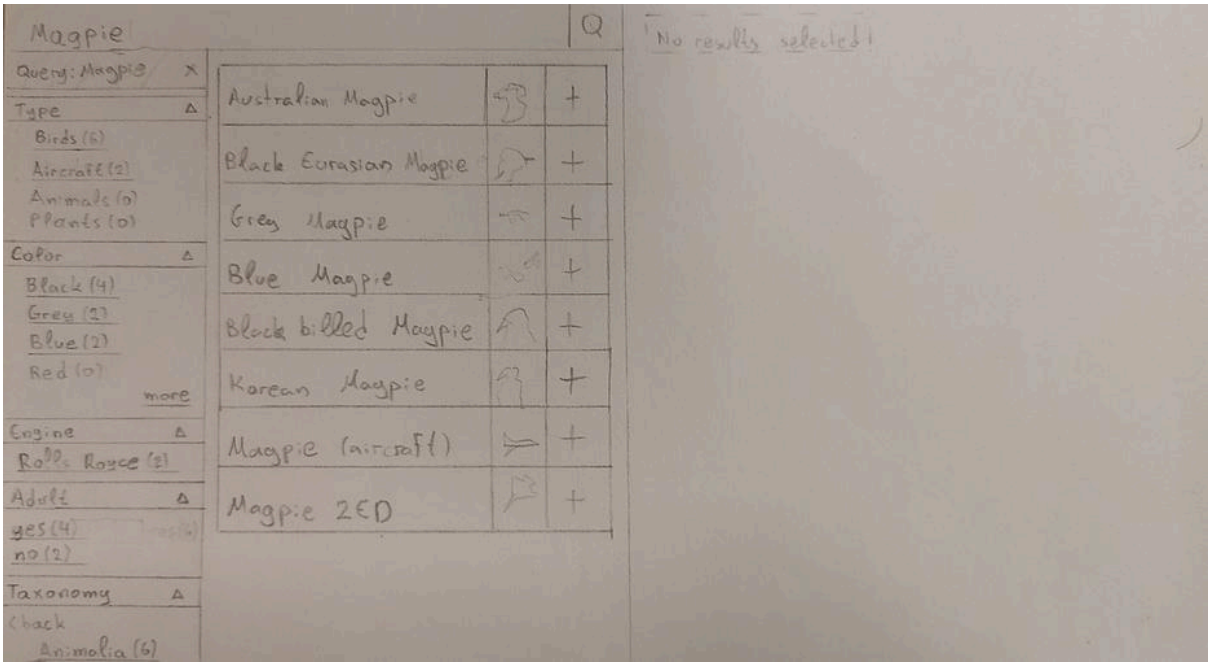


Figure 3.3: Desktop search page storyboard sketch, 3rd iteration

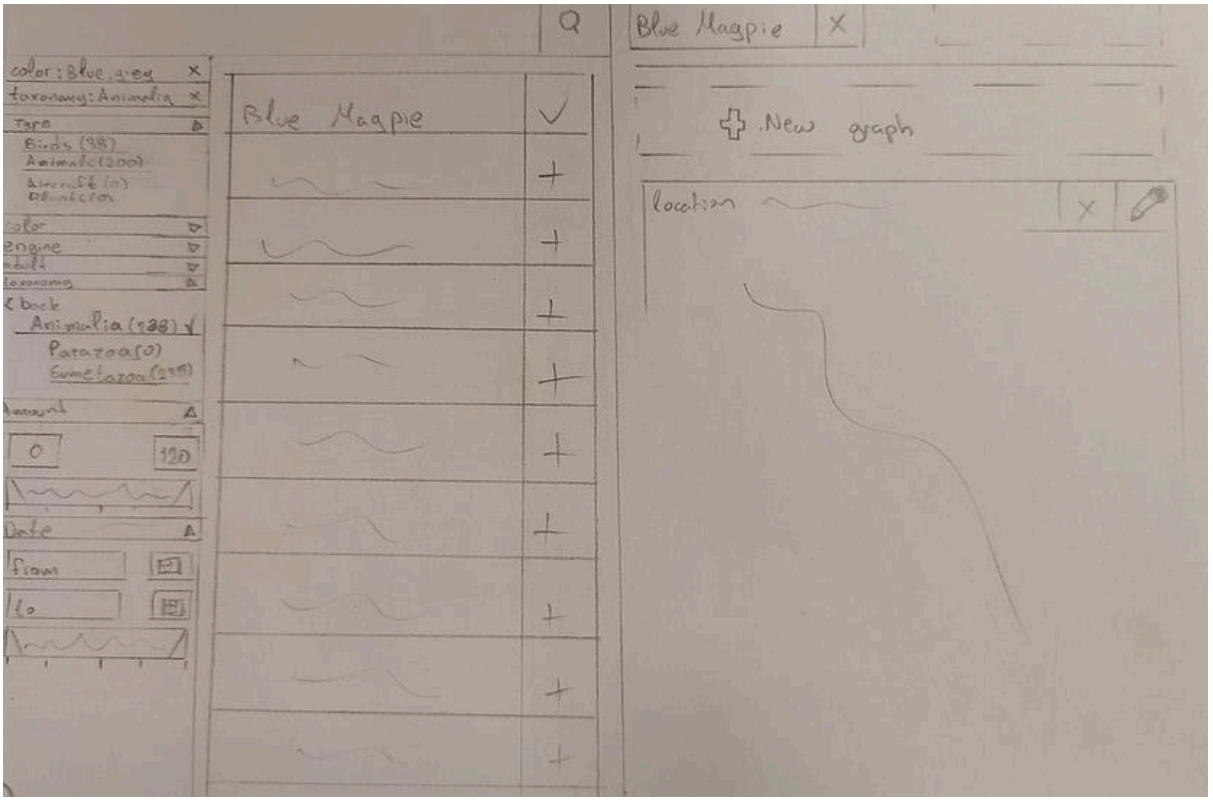


Figure 3.4: Desktop search page storyboard sketch, result selected

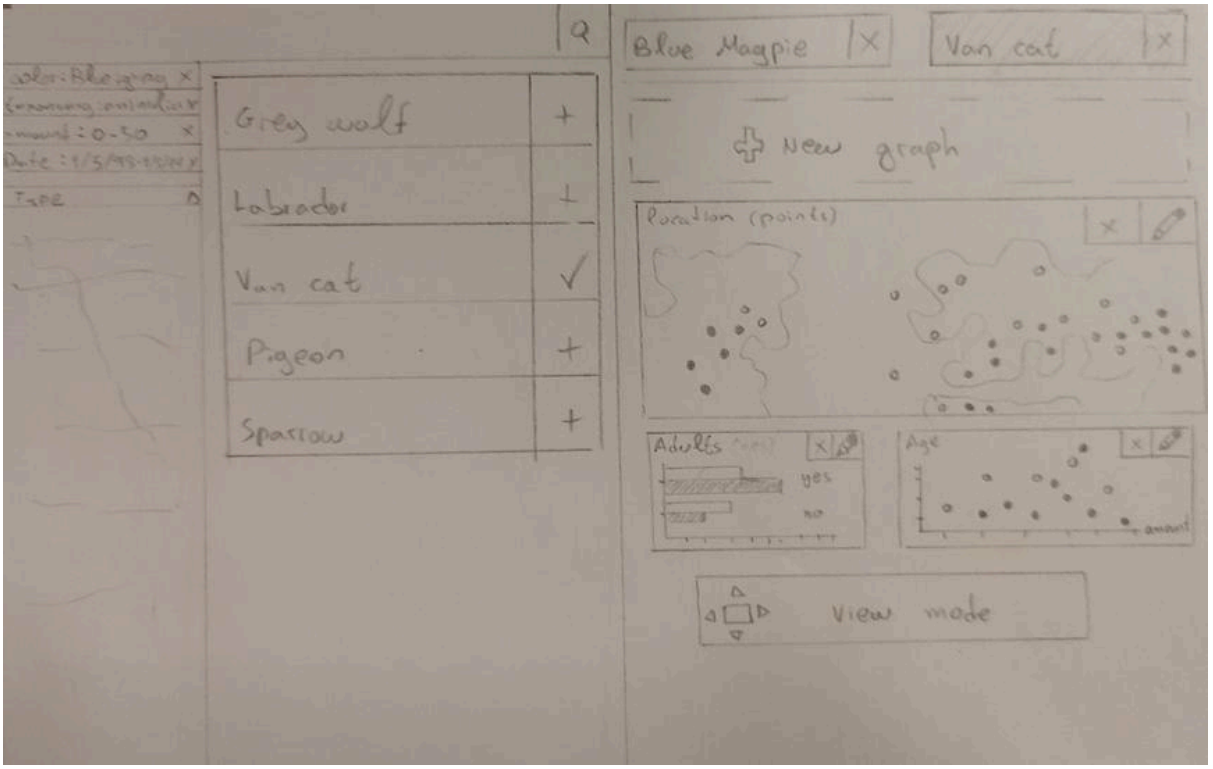


Figure 3.5: Desktop search page storyboard sketch, result comparison

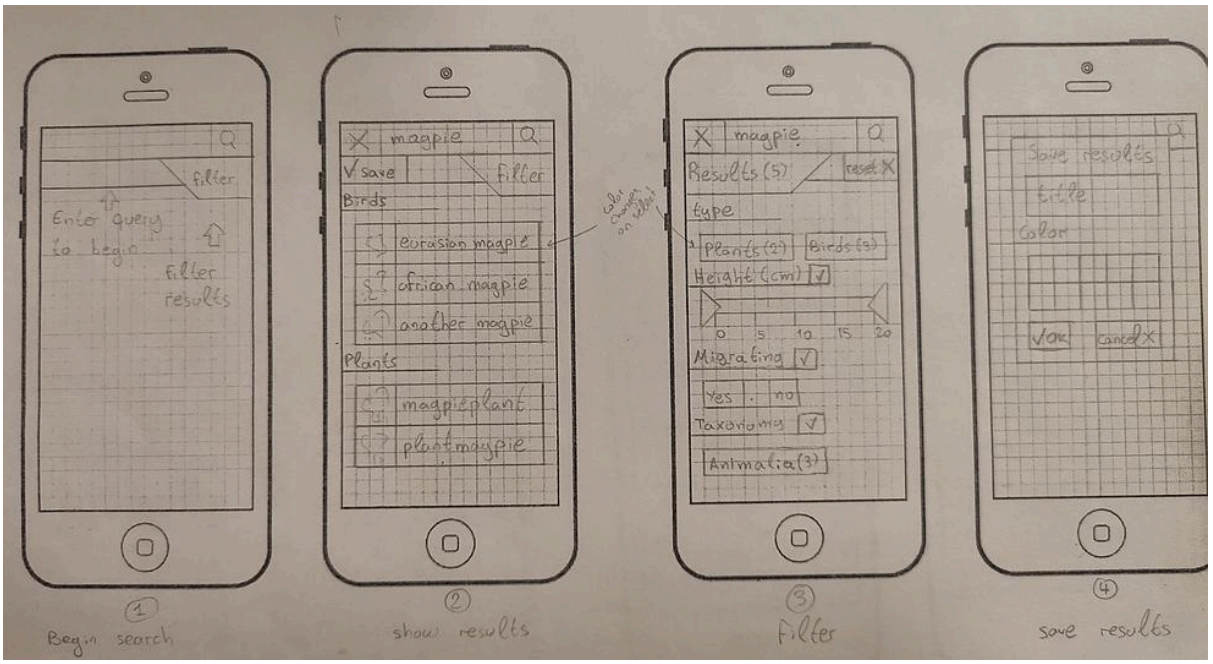


Figure 3.6: Mobile search page storyboard sketch, various numbered states

4. Implementation and Application Features

In this chapter, the features of the app are presented in detail, including observations sharing, trip planning, educational activities, interactive map and navigation, real-time location sharing and team collaboration.

4.1 Technical Requirements

Technical requirements play a crucial role in the development of any software application. They outline the necessary functionalities and attributes that the system must possess in order to successfully meet the needs of its users. This includes both the hardware and software requirements, performance expectations, interoperability demands, and any constraints tied to regulatory standards or best practices.
In the context of our web application, the technical requirements helped to establish the fundamental framework upon which the application was designed and built. These requirements were guided by the unique needs of the users - ranging from park visitors to park management - as well as the specific conditions related to the natural park setting.
The following section presents the technical requirements for our application. It's worth noting that these were not static throughout the development process; instead, they evolved and were refined based on user feedback and iterative testing cycles.

4.1.1 Hardware Requirements

For the **server**, a multi-core processor is essential to efficiently handle multiple simultaneous requests. This ensures that the application remains responsive and performs well under load. At least 8GB of RAM is recommended to facilitate efficient data handling and to support both the databases and server-side applications running concurrently.

In terms of storage, the server should have adequate space to accommodate the application codebase, databases, and any multimedia content such as images or videos. The exact storage requirement will vary depending on the volume and size of this content. Additionally, a reliable and high-speed network connection is critical to maintain the availability and responsiveness of the application at all times.

For the **client side**, users will need a smartphone, tablet, or computer with an active internet connection to access and interact with the application effectively.

4.1.2 Software Requirements

On the **server side**, the application requires a Linux server as the hosting environment. It uses Node.js to run the server-side JavaScript code, with Next.js serving as the React-based web application framework. Strapi.js is used as the content management system (CMS) and for building the REST API. The database is managed using MariaDB, while Redis is employed for managing real-time user groups that involve positional data.

To ensure efficient process management and scalability, PM2 is used as the Node.js process manager, allowing for multiple instances to run across all available CPU cores. Additionally, NGINX acts as a reverse proxy server, handling client requests and directing them appropriately to the server.

On the **client side**, users will access the application through a modern web browser such as Google Chrome, Mozilla Firefox, or Safari. JavaScript must be enabled in the browser, which should also support modern web technologies including HTML5 and CSS3. For desktop users, the application should offer the same features as the mobile version, but with interfaces specifically optimized for larger screens to enhance usability.

4.1.3 Security Requirements

The application must be developed in accordance with secure coding practices to safeguard against common web vulnerabilities. It is essential that user data and passwords are managed and stored securely. On the server side, strong security measures such as firewall configurations and access control policies must be implemented to prevent unauthorized access and defend against potential attacks.

4.2 Functional Requirements

The functional requirements of the web application and servers pertain to the specific tasks and services it should provide to end-users. We outline the primary functionalities expected from the application:

4.2.1 User Account Management

The application needs to offer capabilities for users to create and manage their accounts. This functionality should include secure authentication mechanisms to protect user data.

4.2.2 Trip Planning

The application is required to provide an effective platform for trip planning. Users should be able to define their routes, mark their points of interest, and schedule activities within the

park. The feature should provide the ability to save plans and share them with other users if desired.

4.2.3 Observations Sharing

A crucial function of the application is to allow users to share their observations about the park's flora, fauna, and other environmental features in real-time. This feature should support text entries, photographs, and location tagging, enabling users to create a rich database of observations from their visits.

4.2.4 Real-time Location Sharing

For groups visiting the park, the application needs to offer real-time location sharing functionality. Users should have the option to share their current location with other group members, facilitating coordinated exploration of the park.

4.2.5 Interactive Map and Navigation

The application must provide an interactive map of the park that enables users to navigate easily. This map should display key features of the park, user-defined routes, and points of interest. Navigation tools should be user-friendly and provide accurate location data, facilitating efficient exploration of the park.

4.2.6 Educational Activities

To increase engagement and learning outcomes, the application should offer educational games or activities related to the park's ecosystem. These activities could be quizzes, information hunts, or interactive guides. The games should be engaging, educational, and suitable for a range of age groups.

4.2.7 RESTful Services

For a web application, it's crucial for the server to provide RESTful services to support interactions between its different components and the user's client-side application. These services allow the client-side application to create, read, update, and delete (CRUD) resources on the server, aligning with the REST architectural style.

These services should include:

User services, which allow for user account management operations like account creation, authentication, password resetting, and profile editing.

Observations services, which enable users to submit, edit, and delete their own observations, and view observations submitted by other users.

Trip planning services, enabling users to create, update, and delete their planned routes and activities.

Location services, which allow users to share their current location and view the locations of other users in their group.

Educational activities services, enabling the retrieval of educational games or activities, submission of user answers, and tracking of user progress.
Each service should be secure, reliable, and efficient, ensuring a smooth and enjoyable user experience. The services should also be scalable to accommodate the potential increase in user traffic and interactions over time.

4.3 Technologies used

We employed a range of technologies to achieve the required functionality, performance, and user experience for the web application, paying special attention to their robustness and scalability:

4.3.1 TypeScript

TypeScript, a statically-typed superset of JavaScript, was used extensively in the development of the application. This technology enabled stronger type-checking during development, resulting in code that was more robust, maintainable, and less prone to runtime errors.

4.3.2 CSS3 modules

In creating the visual aesthetics and layout of the web application, CSS3 (Cascading Style Sheets Level 3), a style sheet language used for describing the HTML look and formatting, was utilized extensively.

Moreover, we employed CSS modules, a CSS file in which all class names and animation names are scoped locally by default. This approach, when combined with React, kept our styles encapsulated to specific components, thereby reducing the likelihood of style conflicts across the application.

4.3.3 React

React is a JavaScript library developed by Facebook for creating dynamic and interactive user interfaces. It uses a component-based architecture, which promotes code reusability and the creation of complex UIs from simple and isolated pieces of code. The library

employs a virtual DOM to enhance performance by minimizing direct manipulations of the actual DOM.

In our web application, React was used to construct modular and interactive user interfaces. Each feature of the app, such as the interactive map or the real-time location sharing, was implemented as self-contained React components.

4.3.4 Next.js

Next.js is a popular React framework known for its server-side rendering (SSR) capabilities. Unlike traditional client-side rendering, where the entire application is loaded and rendered in the client's browser, SSR generates the initial application HTML on the server, which improves initial load performance and SEO.

In our web application, we primarily employed Next.js for its SSR capabilities. This approach allowed our application to deliver content to users more quickly, minimizing the time they had to wait for the application to become interactive. In the context of our app, this performance enhancement was vital for providing an excellent user experience, particularly for users with slower internet connections or less powerful devices.

4.3.5 Leaflet

Leaflet, a JavaScript library for interactive maps, was integrated to provide user-friendly, interactive mapping capabilities. It supported various mapping features necessary for the app, such as markers, layers, paths, areas, and event handling.

4.3.6 Node.js

Node.js served as the primary runtime environment for executing server-side JavaScript code. It provided the necessary tools and libraries to build scalable and efficient web servers for the application.

4.3.7 Redis

Redis, an in-memory data structure store, was employed for managing real-time user groups with positional data. It facilitated high-performance operations and provided a Pub/Sub system to allow real-time collaboration between users.

4.3.8 Socket.IO

Socket.IO, is a library that enables real-time bidirectional and event driven communication between web clients and servers. It uses the WebSockets protocol for a low overhead channel, but falls back to long polling if it’s not supported.

4.3.9 MariaDB

MariaDB, an open-source relational database, was chosen for data storage. It offered a robust, secure, and scalable solution for managing the application's data with ACID compliance and complex query support.

4.3.10 Strapi

Strapi, a Node.js-based, open-source headless CMS, was used to manage content and user data, providing a flexible and extendable system for efficient content creation, management, and delivery.

4.3.11 PM2

PM2, a production process manager for Node.js applications, was utilized for application management and automatic restarts, ensuring that the application could run continuously and stably.

4.3.12 Knex.js

Knexjs is a SQL query builder for JavaScript. It provides a flexible and powerful interface for building and executing SQL queries. Knex.js is often used as the query builder for ORMs like Bookshelf.js.

4.3.13 NGINX

Nginx was employed as a reverse proxy server for the application (forwards client requests to appropriate backend servers and returns response to client). It helped manage and direct web traffic and provided additional security features.

4.3.14 GraphQL

To complement the base RESTful architecture, GraphQL was integrated into the application to provide efficient and flexible data querying and manipulation. This technology allowed clients to request specific data as needed, reducing unnecessary data transfers and improving performance.

4.3.15 Git

Version control is an essential part of modern software development, and for this project, we used Git. Git allowed us to effectively track and manage changes to the project, making it possible to try out new features or changes in separate branches without disturbing the main codebase.

This capability was particularly useful in a collaborative setting, as it facilitated simultaneous development on different parts of the application. Additionally, through the use of Git, we could easily revert changes if something went wrong, providing a safety net for the development process. Lastly, Git's collaboration features were critical for remote team

collaboration, particularly when used in conjunction with a web-based hosting service like GitHub or GitLab.

4.3.16 Cloudflare CDN

Used for content delivery, improving the performance and security of applications by caching content and providing DDoS protection.

4.4 Frontend and Backend Development

This section discusses the development process of the frontend user interface and the backend system, including the choice of technologies, development environments, and key features implemented. It also includes a description of how user input and server responses are handled.

The architecture of the system is constructed to support the web application effectively and efficiently, ensuring robust performance and seamless user interactions. This architecture is hosted on a Virtual Private Server (VPS) running Ubuntu, a widely adopted Linux distribution known for its stability and security.

4.4.1 Server Hosting and Application Components

The VPS hosts several critical services and servers, each tailored to manage specific aspects of the web application:

App Server: The App Server leverages Next.js, a React and Nodejs based framework designed for server-side rendering (SSR). This architectural choice optimizes performance by pre-rendering pages on the server and delivering them as static HTML to the client's browser. When a user requests a page, Next.js processes this request, generates the corresponding HTML, and sends it to the client, ensuring faster initial load times and improved SEO outcomes. The SSR capability also allows for dynamic content to be served efficiently, based on user interactions or other triggers, enhancing the overall user experience.

Location Server: This node.js server is responsible for managing real-time communication and user interactions within the application. Utilizing Socket.IO, it establishes WebSocket connections that facilitate instant data exchange between users. Users' locations are broadcasted in real-time to all members of the group, enabling features such as live navigation and team collaboration. The server interacts with Redis to manage the real-time location data of users. It uses Redis Pub/Sub to push key changes (like location updates or key expirations) to clients in real-time.

API/CMS Server: The API/CMS Server is built on Strapi.js, a flexible nodejs based headless CMS that serves as the backend for content management. This server provides RESTful and GraphQL APIs, allowing the front-end application to retrieve and manage data

seamlessly. This server handles requests for data related to park features, educational resources, user-generated content, and any other information the app needs to deliver. It also provides robust content management capabilities tailored for handling dynamic data within the web application, as it allows park administrators to easily create, edit, and manage content.

MariaDB Server: This relational database management system (RDBMS) is employed for data persistence, ensuring reliable storage and retrieval of structured data. MariaDB is robust, ACID-compliant, and supports complex queries, making it an excellent choice for handling the intricate relationships among various data entities within the application. It is configured to interact solely with the API/CMS Server, thereby enhancing security and ensuring that all database transactions are managed via validated API calls.

Redis Server: The Redis Server is an in-memory data structure store that optimizes the performance of the application through caching and real-time data processing. It is specifically used for managing session data and facilitating the real-time collaboration features of the application. Similar to the MariaDB server, the Redis server is also not exposed to the internet, with access limited to the internal Location Server.

4.4.2 Process Management with PM2

The operational reliability of the App Server, Location Server, and API/CMS Server is maintained through the use of PM2, a production-grade Node.js process manager. PM2 oversees the execution of these Node.js services, providing essential features such as automatic process restarts on failure, monitoring, and logging capabilities. By keeping the servers operational and responsive, PM2 plays a crucial role in enhancing the overall stability of the web application.

4.4.3 Reverse Proxy with NGINX

To streamline communication between clients and the various backend services, NGINX serves as a reverse proxy server. It acts as an intermediary, forwarding incoming requests to the appropriate internal server based on the requested subdomain. For instance, requests directed at the main subdomain are routed to the App Server, while requests for administrative or API access are directed to the API/CMS Server. By acting as a gateway, NGINX helps in optimizing response times and ensuring secure connections through SSL/TLS encryption.

4.4.4 Internal Communication and Data Handling

The various servers in this architecture communicate through well-defined protocols and methodologies to ensure efficient data handling and seamless interactions.

The **App Server** communicates with the **API/CMS Server** primarily through a GraphQL API. This architecture allows the App Server to fetch, create, and manipulate data dynamically. When a user requests data, such as park information or user-generated content, the App

Server checks if a page has already been pre-rendered. If yes, it is sent immediately to the user and then revalidates for any changes. Otherwise sends a request to the **API/CMS Server**. The response is sent back to the App Server, where it is rendered server-side. Next.js processes the response to dynamically update the content that will be served to the client. The App Server then sends the pre-rendered HTML, enriched with the retrieved data, back to the client's browser for display. This pre-rendering ensures faster page load times and better SEO, compared to client-side rendering.

The **API/CMS Server** interacts with the **MariaDB Server** exclusively through Knex.js, a SQL query builder for Node.js that facilitates the construction of complex SQL queries while maintaining a clean and intuitive syntax. This approach ensures that all database interactions are handled safely and efficiently, minimizing the risk of SQL injection attacks while optimizing query performance.

The **Location Server** communicates with clients via Socket.IO, which enables real-time, bidirectional communication. This library abstracts the complexities of WebSocket connections and fallbacks, allowing developers to implement real-time features with ease. By establishing persistent connections, it allows users to receive updates instantly, such as other users' locations, fostering an engaging and collaborative experience in the park environment. Two Redis clients are utilized: one for storing user data (repo) and another for subscribing to key events (sub). Redis is configured to handle keyspace notifications for events like key expiration, which ensures that outdated user locations are automatically cleaned up and clients are notified in real-time.

- **Repository Client**: This client handles writing and retrieving real-time data such as user locations, group status, and other session data. The locations of users are stored with an expiration time, ensuring that the system only tracks active users.

- **Subscription Client**: This client listens for keyspace notifications. When a user’s data is updated (e.g., location), Redis publishes a key event, which is then picked up by the subscription client. The client pushes the updated data to the Location Server, which emits the data to sockets of connected clients.

4.4.5 Security Considerations

Security is a paramount concern in any web application, particularly those that handle user data and provide real-time interactions. The application implements a comprehensive, multi-layered security strategy:

Authentication Security: The system employs bcrypt for password hashing with automatically generated salts, protecting user credentials against rainbow table and brute force attacks. Each user password is combined with a unique random salt before hashing, ensuring that identical passwords still produce different hash values in the database. This approach, integrated with Strapi's authentication framework, provides robust protection for user accounts without requiring manual security implementation.

Database Security: The MariaDB Server is configured to accept connections exclusively from the API/CMS Server, and the Redis Server is configured to accept connections exclusively from the Pub/Sub Socket Server, ensuring that direct access from external sources is strictly prohibited. This design minimizes the attack surface by ensuring that the database is shielded from direct internet exposure. Furthermore, sensitive SQL operations are protected by parameterized queries via Knex.js, thereby preventing SQL injection vulnerabilities.

Network Security: Communication channels throughout the application are secured using HTTPS via Cloudflare's authenticated origin pull, ensuring that all data transmission is encrypted. NGINX is configured with rate limiting to mitigate denial-of-service attacks, restricting request frequencies based on client IP addresses while whitelisting internal services.

Real-time Communication Security: The communication channels established via Socket.IO are secured using HTTPS, ensuring that data transmitted between the server and clients is encrypted. This security measure is crucial for maintaining the confidentiality and integrity of user data during real-time interactions. Additionally, real-time features like location sharing incorporate privacy controls that allow users to explicitly enable sharing within specified groups, while automated key expiration in Redis ensures that outdated location data is promptly purged from the system.

This defense-in-depth approach, where multiple security controls work in concert, provides comprehensive protection for both user data and system integrity across all components of the application.

4.4.6 Monitoring

Ongoing monitoring is vital to maintain application performance and ensure a high availability of services.

Monitoring: The application can be monitored using PM2’s built-in monitoring tools, which provide insights into system performance, resource usage, and any potential errors. Additionally, logs can be collected and analyzed to identify trends and areas for improvement. Alerts can be configured to notify the development team of any anomalies, enabling proactive measures to be taken to maintain application stability.

4.4.7 User Interface and Experience

The user interface (UI) is a critical aspect of the web application, as it dictates how users interact with the various features and functionalities provided.

UI Design Principles: The UI was developed with a focus on usability and accessibility, ensuring that all users can navigate the application easily. Interactive elements are designed to be intuitive, with clear visual cues guiding users through their interactions.

Responsive Design: The web application employs responsive design principles, allowing it to adapt seamlessly across various devices and screen sizes. This ensures that visitors can access the application on their mobile phones, tablets, or desktops, providing a consistent experience regardless of the platform.

The overall architecture of the web application is designed to provide a scalable, secure, and efficient platform for enhancing the visitor experience in natural parks. By leveraging modern technologies and best practices in web development, the application aims to foster an engaging and collaborative environment for users, while also supporting the operational needs of park management.

4.5 Database Design

This section provides an overview of the database architecture, detailing how data is organized, stored, and interconnected to support various features of the application, including user interactions, content management, and real-time collaboration.

The Entity-Relationship (ER) diagram presented in this section illustrates the core entities within the database and how they relate to one another. Each entity, such as users, park features, observations, trips, represents a fundamental component of the application.

4.5.1 Taxon, Observation, Metadata & Metavalues (Taxonomic Data Model)

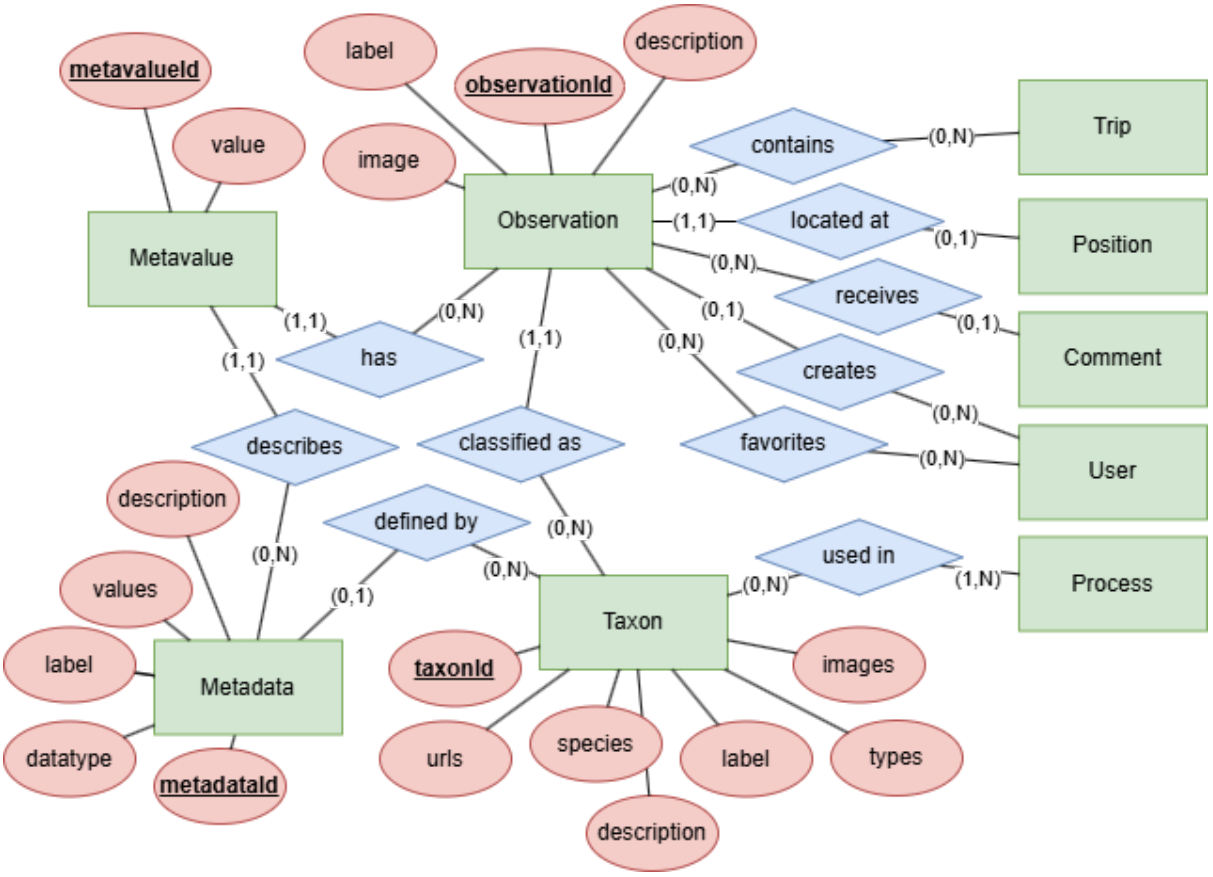


Figure 4.1: ER sub-diagram illustrating the taxonomic data model with interconnections between taxon, observation, metadata, and metavalue entities

The taxonomic system in the database is designed to flexibly capture and organize observations of flora and fauna while maintaining scientific accuracy and supporting dynamic metadata collection. The interaction between Taxon, Observation, Metadata, and Metavalues creates a flexible yet structured system for recording natural observations.

Taxon

The Taxon entity serves as the taxonomic authority and classifier in the system. It defines valid species that can be observed in the park, establishes which metadata types are relevant for each species, groups organisms by type (plants/animals) to enable filtered searching, stores scientific nomenclature and common names, and finally provides descriptive information about species characteristics

For example, a "Fennel" taxon would define that observations of this plant should collect metadata about flowering season, growth habits, and uses.

Observation

Observations represent actual sightings or recordings made by users in the park. They link user-generated content to specific taxa, record spatial and temporal data about the sighting, associate relevant metadata values based on the taxon's requirements, enable photo attachments and descriptions, and track who created the observation and who has favorited it.

An observation of a fennel plant, for instance, would record its location, when it was seen, what stage of growth it was in, and any photos taken by the user.

Metadata Types (Metadata)

Metadata types define the categories of information that can be collected about different taxa. They establish the valid attributes that can be recorded for observations, define the data type expected (numeric, list, text, etc.), provide validation rules for entered values, support multilingual descriptions and labels, and can be shared across multiple taxa

Examples include "flowering season" for plants or "body length" for animals.

Metadata Values (Metavalues)

Metavalues store the actual data recorded for specific observations based on the metadata types. They link specific values to individual observations, validate entered data against the metadata type rules, enable filtering and searching of observations, support scientific analysis and reporting, and maintain data integrity through foreign key relationships

For a fennel observation, metavalues might record "July" for flowering season and "Perennial" for growth habit.

Interaction Flow

1. Taxon Definition: Park administrators define taxa in the system, then each taxon is associated with relevant metadata types, and finally valid values or ranges are established for each metadata type.

2. Observation Creation: User selects a taxon when creating an observation, then the system dynamically loads relevant metadata fields based on taxon, and finally the user fills in metadata values along with location and description.

3. Data Validation: System validates entered metadata values against type definitions, ensures scientific accuracy and data consistency and maintains referential integrity across the database.

4. Query and Analysis: Enables complex queries across observations, supports filtering by metadata values and facilitates scientific research and trend analysis.

Design Benefits

1. Flexibility: New taxa can be added without database schema changes. Metadata types can be modified or added as needed. Supports various types of observations and data collection.

2. Data Integrity: Enforces valid relationships between observations and taxa. Ensures metadata values match defined types. Maintains scientific accuracy of recorded data.

3. Scalability: Efficiently handles growing numbers of observations. Supports addition of new metadata types. Enables complex queries without performance degradation.

4. Research Support: Facilitates scientific analysis of observations. Enables tracking of species distribution and behavior. Supports environmental monitoring and research.

This design creates a robust foundation for citizen science and environmental education while maintaining data quality and scientific validity. It balances user accessibility with research requirements, making it suitable for both casual park visitors and serious researchers.

4.5.2 Activity

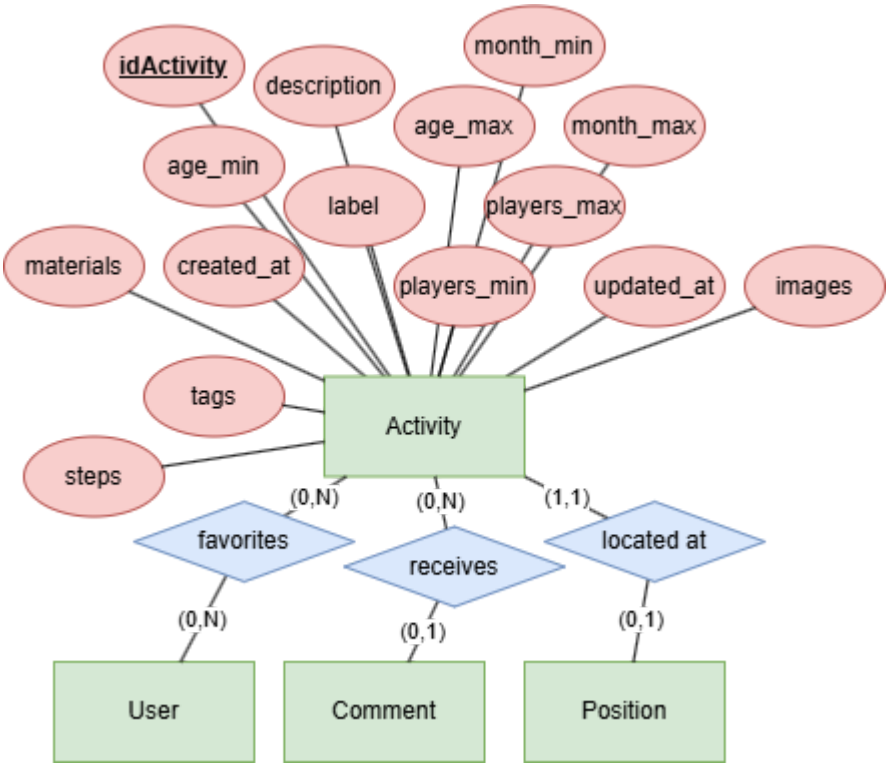


Figure 4.2: Entity-relationship sub-diagram depicting the Activity entity and its associations, showing the data structure for educational experiences in the park system

Activities represent educational experiences and games that can be undertaken in the park. They define educational content and interactive experiences, set parameters for participation (age ranges, group sizes), specify seasonal availability through month ranges, support multiple simultaneous participants, track completion and engagement metrics

4.5.3 Trip

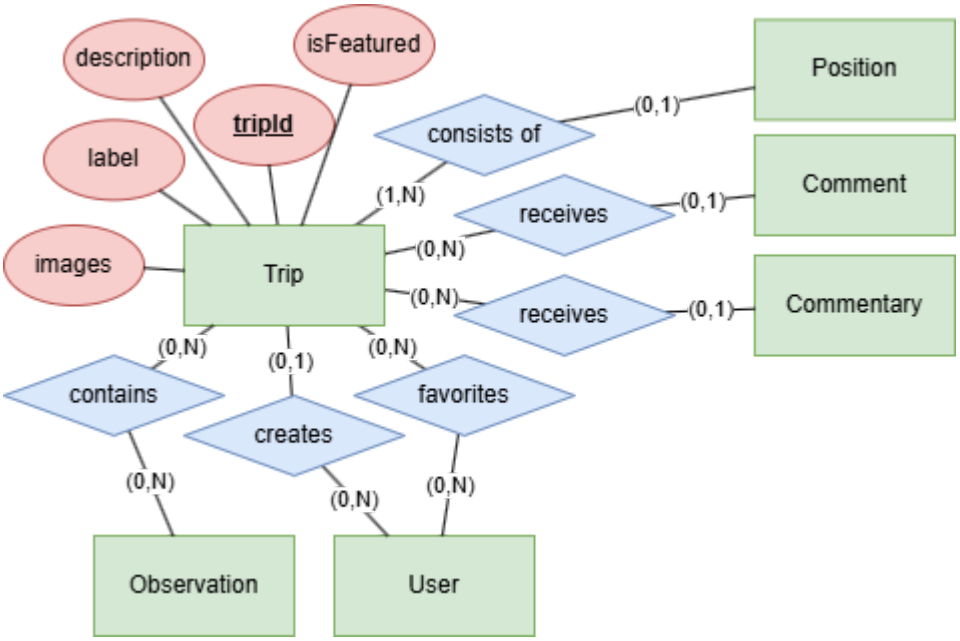


Figure 4.3: Entity-relationship sub-diagram representing the Trip entity, demonstrating the structure for visitor-created park routes

Trips serve as user-created guides through the park. They store sequential path coordinates for navigation, link to relevant observations along the route, include descriptive content and recommendations, track popularity through user favorites, support featured status for official or high-quality routes.

4.5.4 User

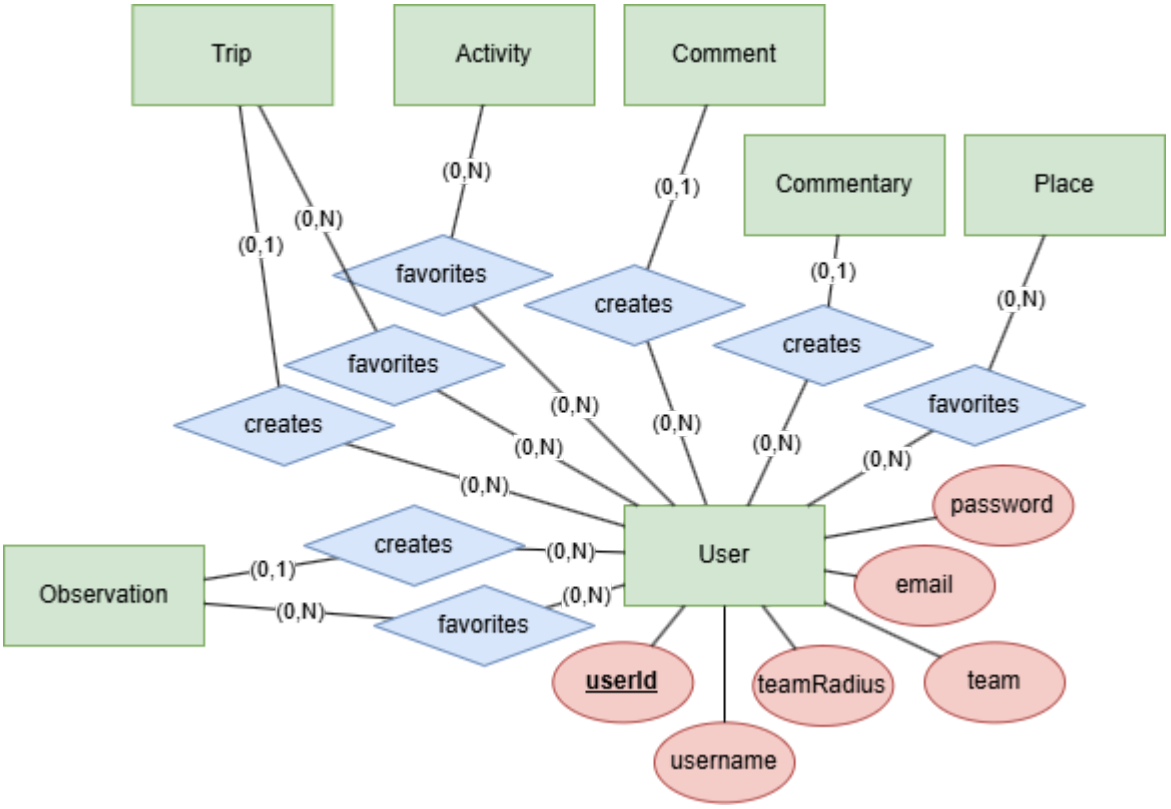


Figure 4.4: Entity-relationship sub-diagram showing the User entity and its relationships with user-generated content

Users are the core actors in the system. They maintain personal profiles and preferences, create and manage content (observations, trips, comments), participate in group activities and real-time location sharing, build reputation through contributions and engagement.

4.5.5 Park

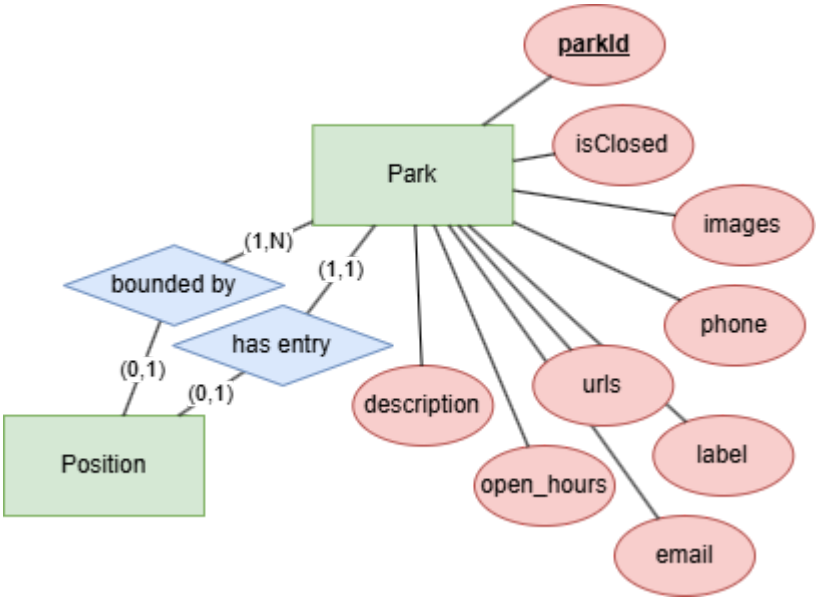


Figure 4.5: Entity-relationship sub-diagram depicting the Activity entity and its associations, showing the data structure for educational experiences in the park system

The Park entity serves as the root container for all location-based content. It defines park boundaries and zones, manages operational information (hours, contact details), coordinates facility and service availability, supports multiple language content.

4.5.6 Position

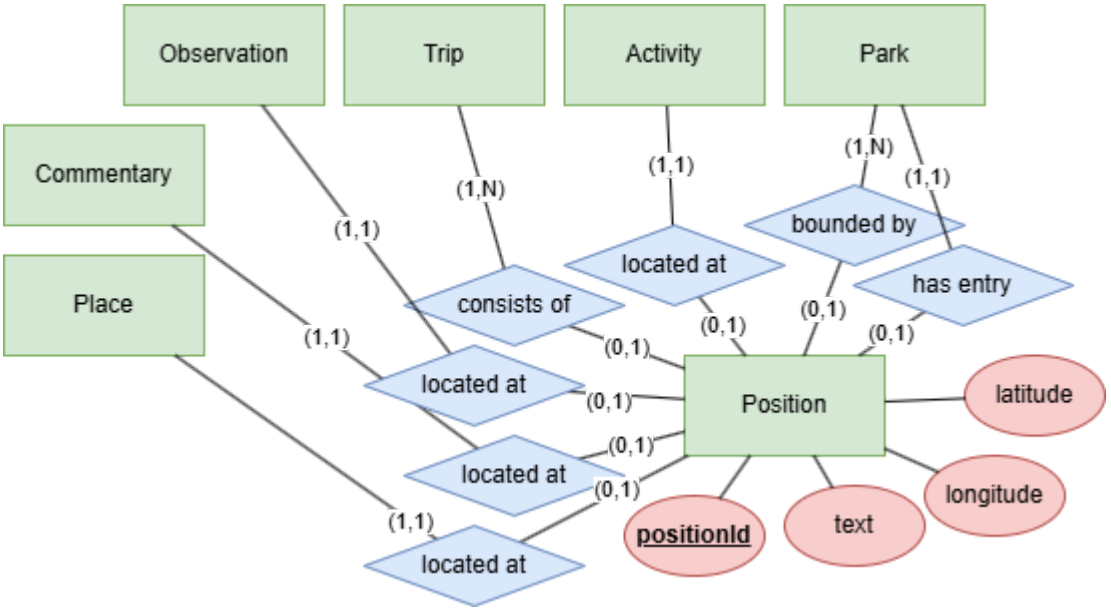


Figure 4.6: Entity-relationship sub-diagram representing the Position entity and its relationships with various location-dependent entities in the system

Positions track spatial information throughout the system. They store coordinates for various entities, support different geometric types (points, lines, polygons), enable real-time location updates, facilitate spatial queries and proximity searches.

4.5.7 Process

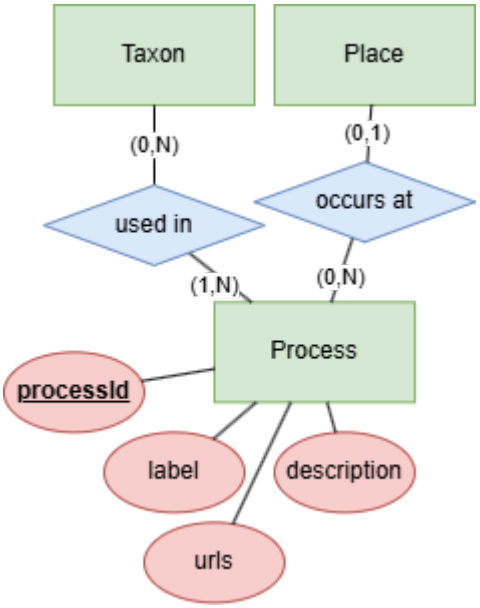


Figure 4.7: Entity-relationship sub-diagram illustrating the Process entity and its connections to both Place and Taxon entities

Processes represent specific ways humans utilize or have historically utilized plants and animals. They document traditional and modern usage methods, describe step-by-step procedures for resource utilization, record cultural and historical significance, track seasonal timing for harvesting or processing, store safety considerations and best practices.

Examples include olive oil production from park's olive trees, traditional medicine preparations from herbs, natural dye creation from plants, food preservation techniques and traditional craft making from plant materials.

4.5.8 Place

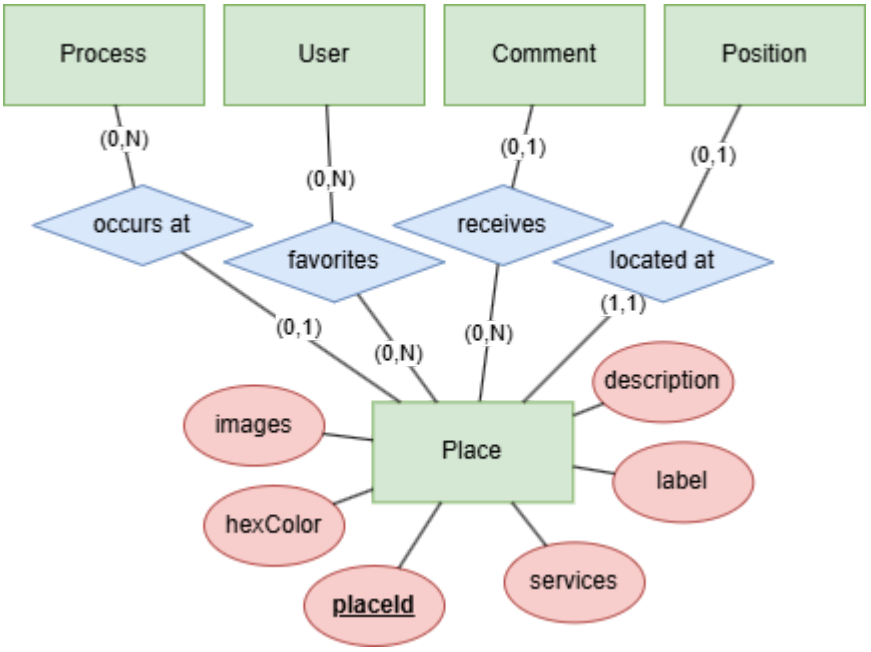


Figure 4.8: Entity-relationship sub-diagram depicting the Place entity with its associations and attributes

Places represent distinct locations or areas in the park. They define points of interest and facilities, manage area boundaries and paths, support different location types (points, areas, routes), include descriptive information and usage guidelines.

4.5.9 Complete ER diagram

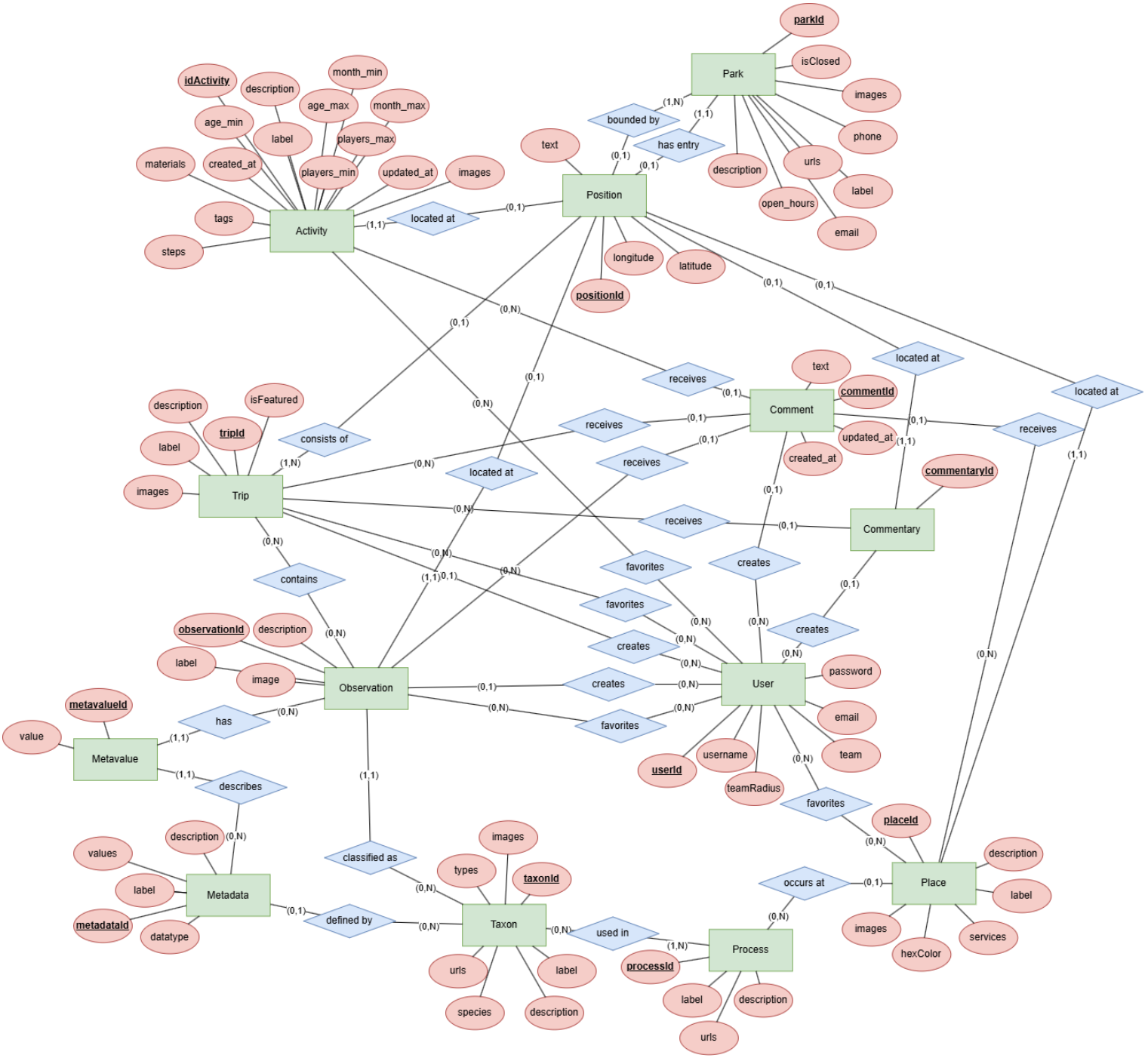


Figure 4.9: Complete entity-relationship diagram of the park management system illustrating all entities and their interrelationships

4.6 Graphical User Interface

4.6.1 User Interface

The page is divided into two parts. The map, and the data section overlay.
This design pattern was chosen because of most users familiarity with google maps, the most recognizable web mapping service.
Both parts interact with one another, changes in the data section update the map, and clicking on a map feature brings up a popup that leads to new information in the data section.
On the top right, there is the account button that allows the user to login, register, update account data & logout.
On the top left, whenever the user has opened a page other than the main one, there are crumbs that guide the user through the website's page hierarchy and back to the homepage.

Map

The user can move around the map using his mouse or fingers depending on the device, but there are also zoom buttons and a layer menu that allows the user to remove or add map metadata types. Users can click on any of the displayed points, areas, or paths, to bring up a popup with its title and the related image. Clicking on the popup loads the related page on the overlay sheet.

Data Section Overlay

The data section overlay scrolls up and down, similarly to a more traditional webpage, when the user scrolls using the mouse or fingers (even on the map). Moving the section recenters the map, keeping related information visible. Additionally, there's a button on the top of the section that when clicked serves the similar function of scrolling the section in and out of view.

4.6.2 Pages

Main Page

The main page's map displays the user's location, and the park's main features, including areas/points of interest (for example buildings, hygiene or service booths) and the main paths. The data section displays the park title along with contact links and other information. Following that is a list of buttons for each type of park data: animals, plants, activities, places & trips that leads to the appropriate search page.
Finally, if the user is logged in, there's a button list for data creation: new observation, and new trip.

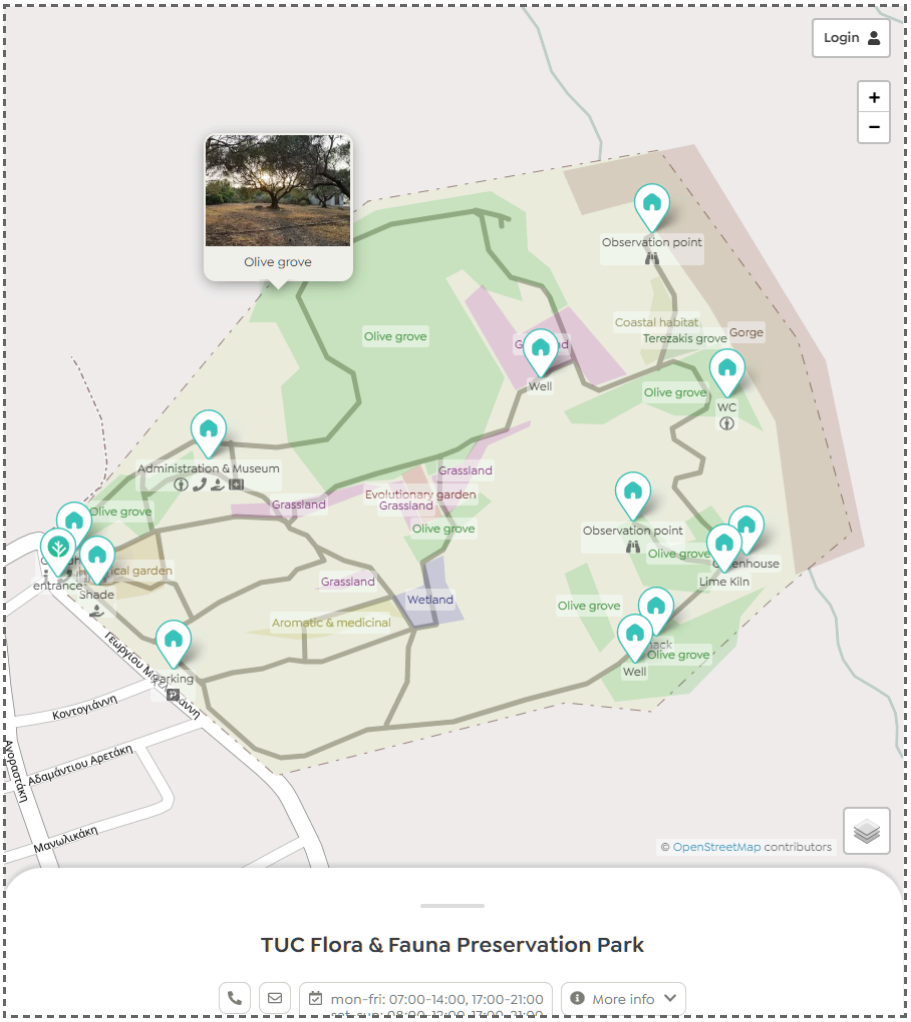


Figure 4.10: Map section of the app UI

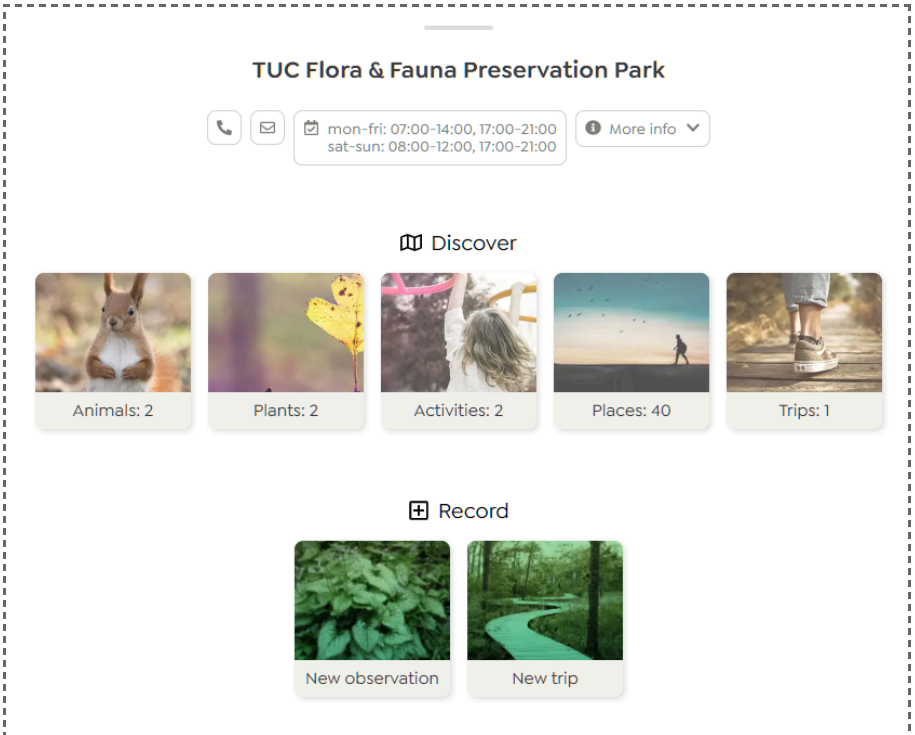


Figure 4.11: Data section overlay of the main page UI

Account Page

By clicking the “account” button on the top right of the screen, the user enters the account page, where he can create a new account or login into an existing one.
If the user is already logged in, the account page instead you can create/join a new group to take part in with other users and monitor their whereabouts, as well as a radius to get informed if any user gets too far away from you (useful for children).

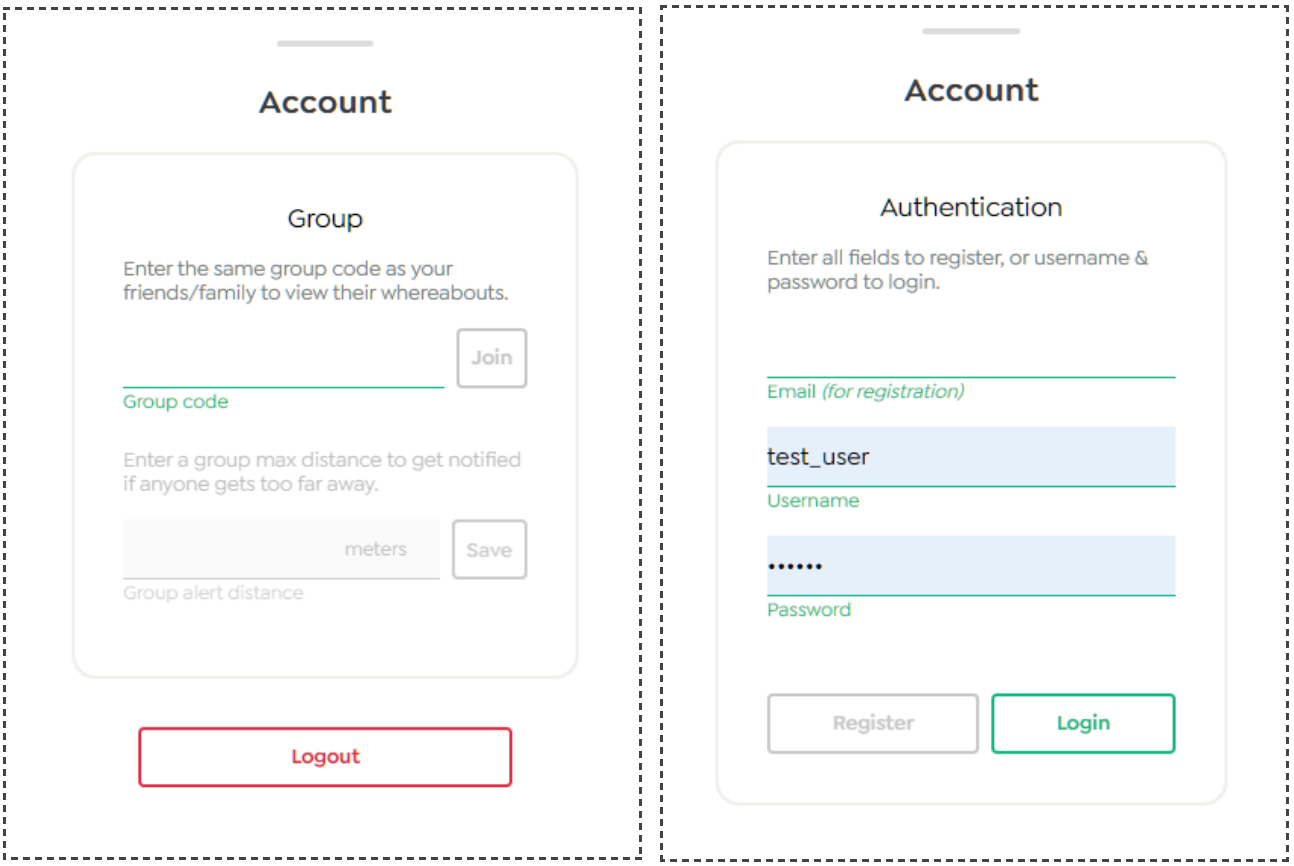


Figure 4.12 (left): Data section overlay of the account page UI. User is logged in
Figure 4.13 (right): Data section overlay of the account page UI. User is logged out

Discover Page

From the main page, by clicking one of the “discover” buttons, the user loads the appropriate search page. The search page contains a text search box, and filters displayed as a series of pills. Each pill expands into a drop down list on click, and displays any selection/s made in closed form. All user interactions update the results and the map automatically.

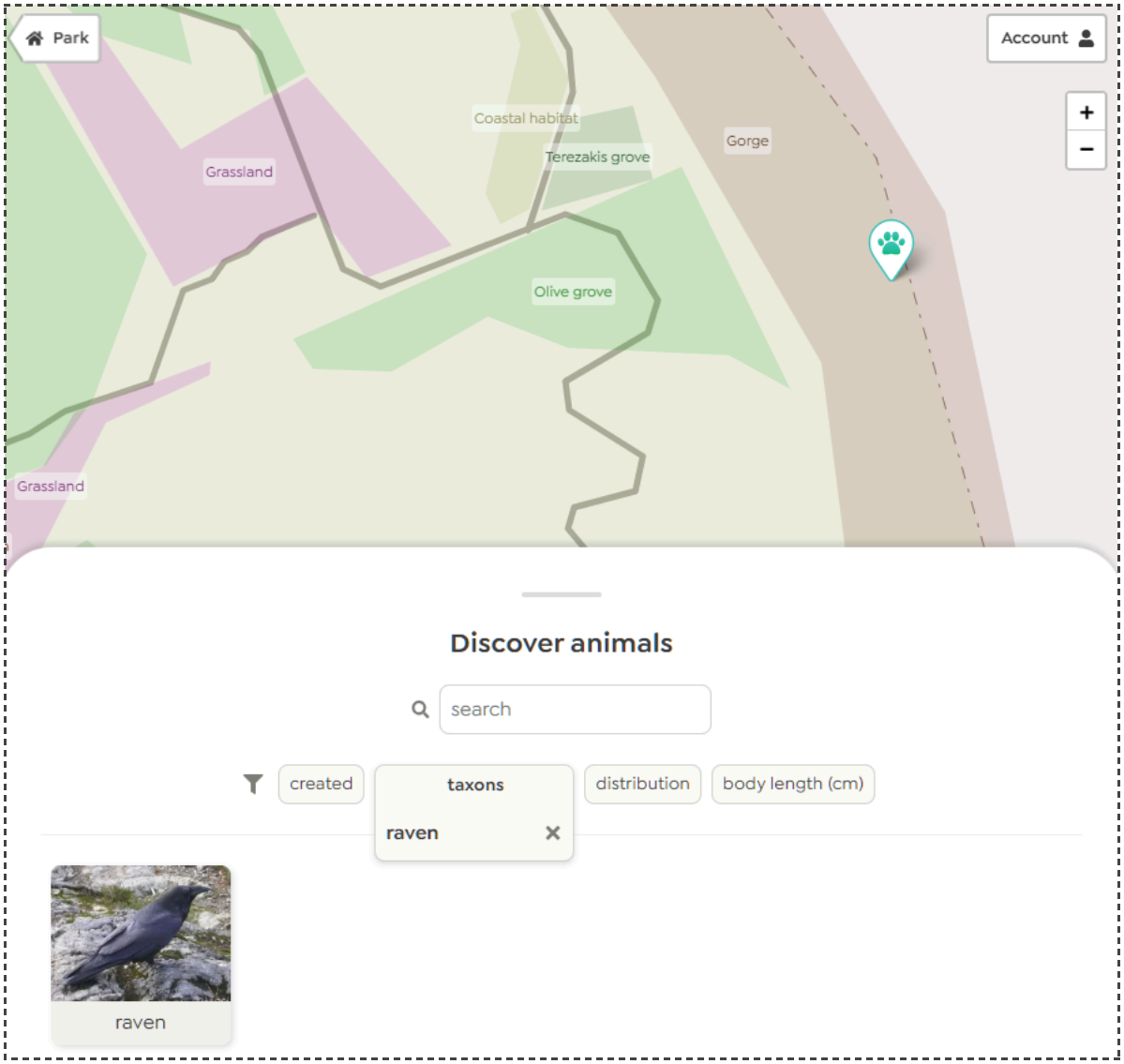


Figure 4.14: Discover page UI. User has filtered the results through a facet

View Page

By clicking a result in the discover page, or an open popup in the map, the user opens the view page. There, information on the observation/trip/activity/place can be seen, including images, a title, description, and other metadata. An edit button is clickable on the upper left corner of the overlay sheet, if the user is the creator of the observation or trip. Furthermore, if the user is logged in, a favorite button is displayed on the upper right, and new comments can be added at the end of the page.

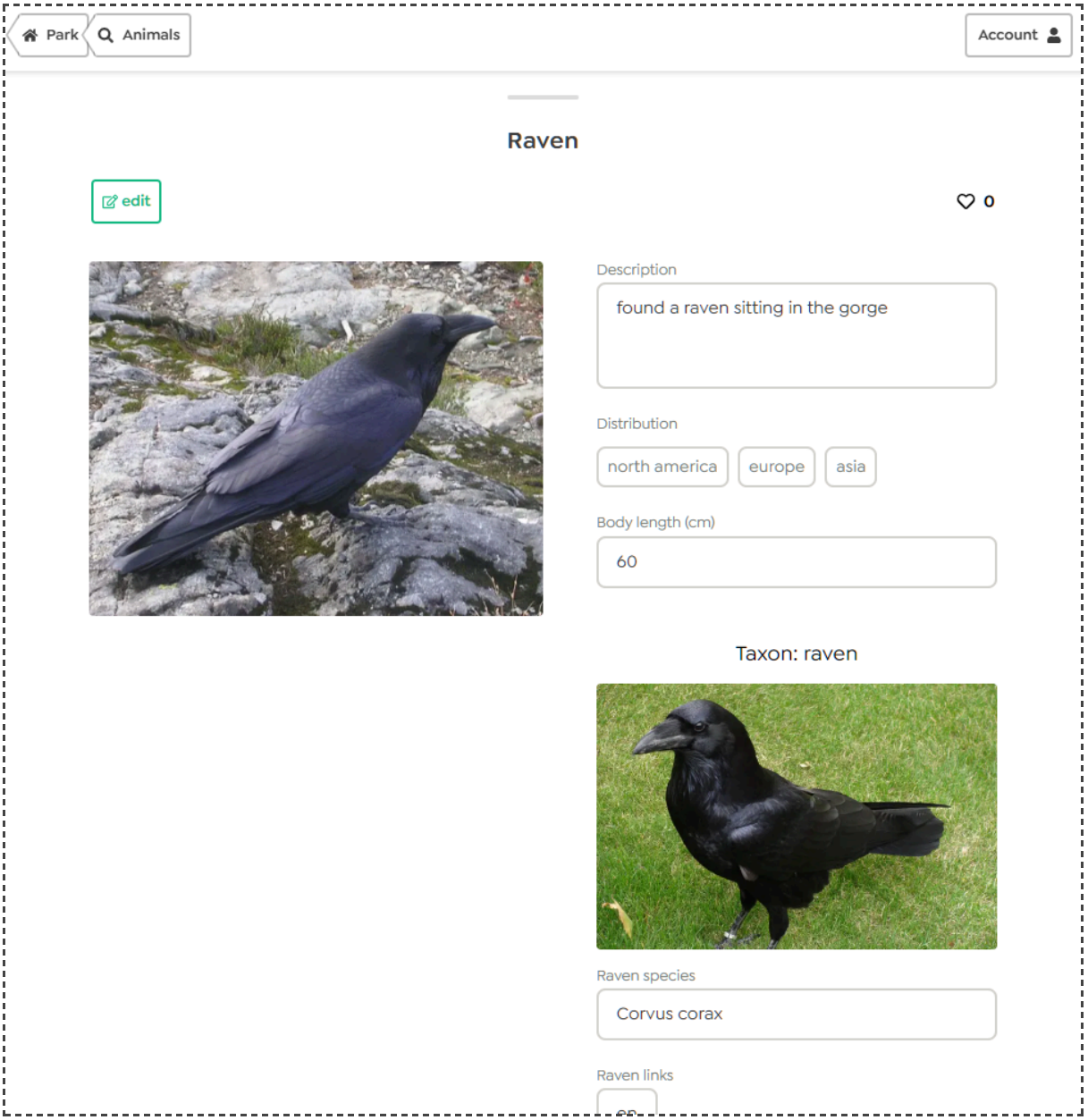


Figure 4.15: Data section overlay of the view observation page UI (Note that the user is the creator of this observation and so an edit button is also displayed)

Edit Page

By clicking the edit button in the view page of an observation or trip (only visible if the user is the creator), the edit page is loaded. Here the user can update any relevant metadata, including the geolocation, and either upload the changes or reset back through the relevant buttons on the upper left of the overlay sheet.

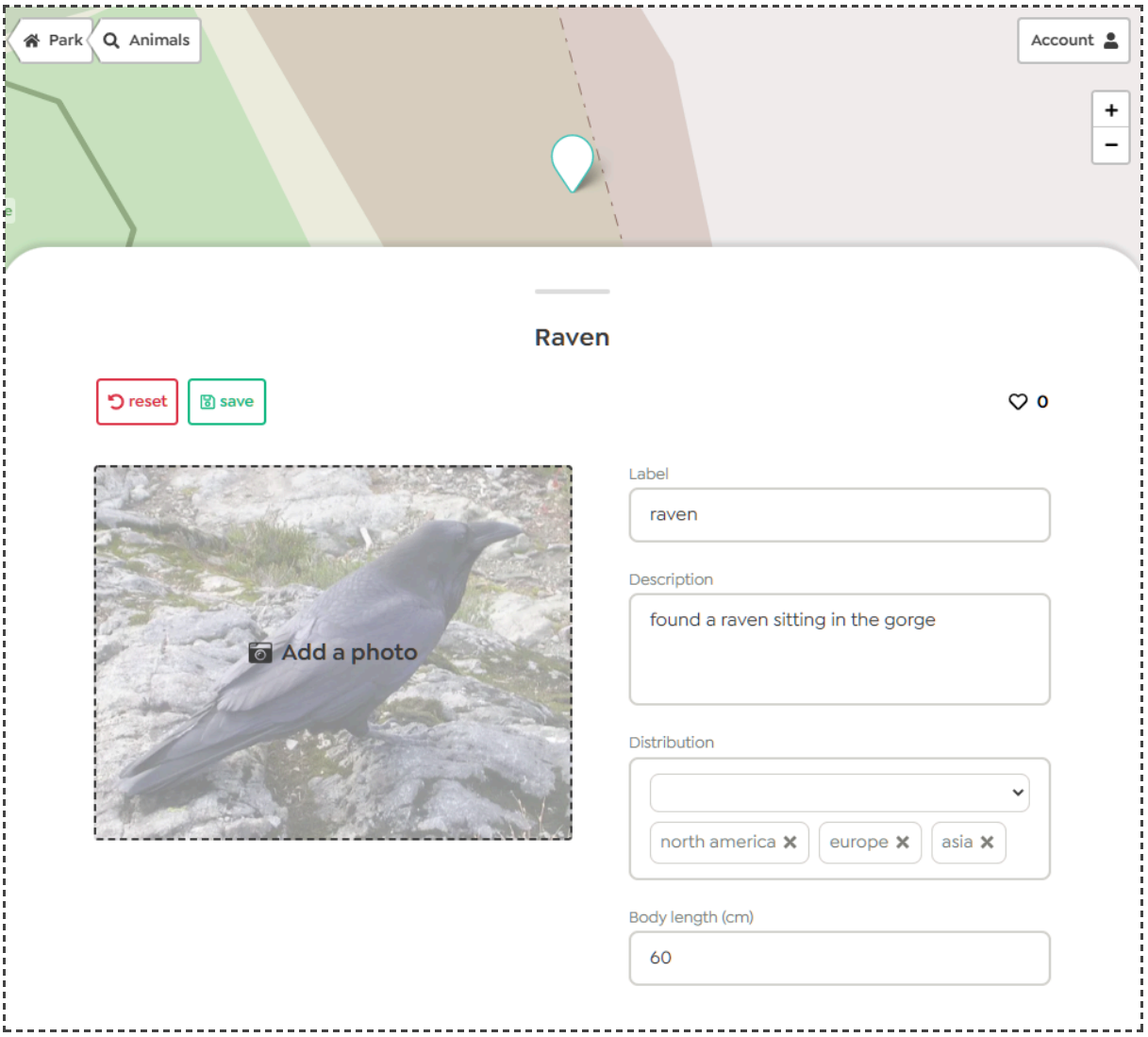


Figure 4.16: Edit observation page UI

Record Page

From the main page, by clicking one of the “record” buttons (new observation or new trip), the user can create a new observation or trip. He can input any relevant metadata, including the geolocation, and either upload the changes through the save button on the upper left of the overlay sheet. The allowed metadata types for observations are generated from the chosen taxon.

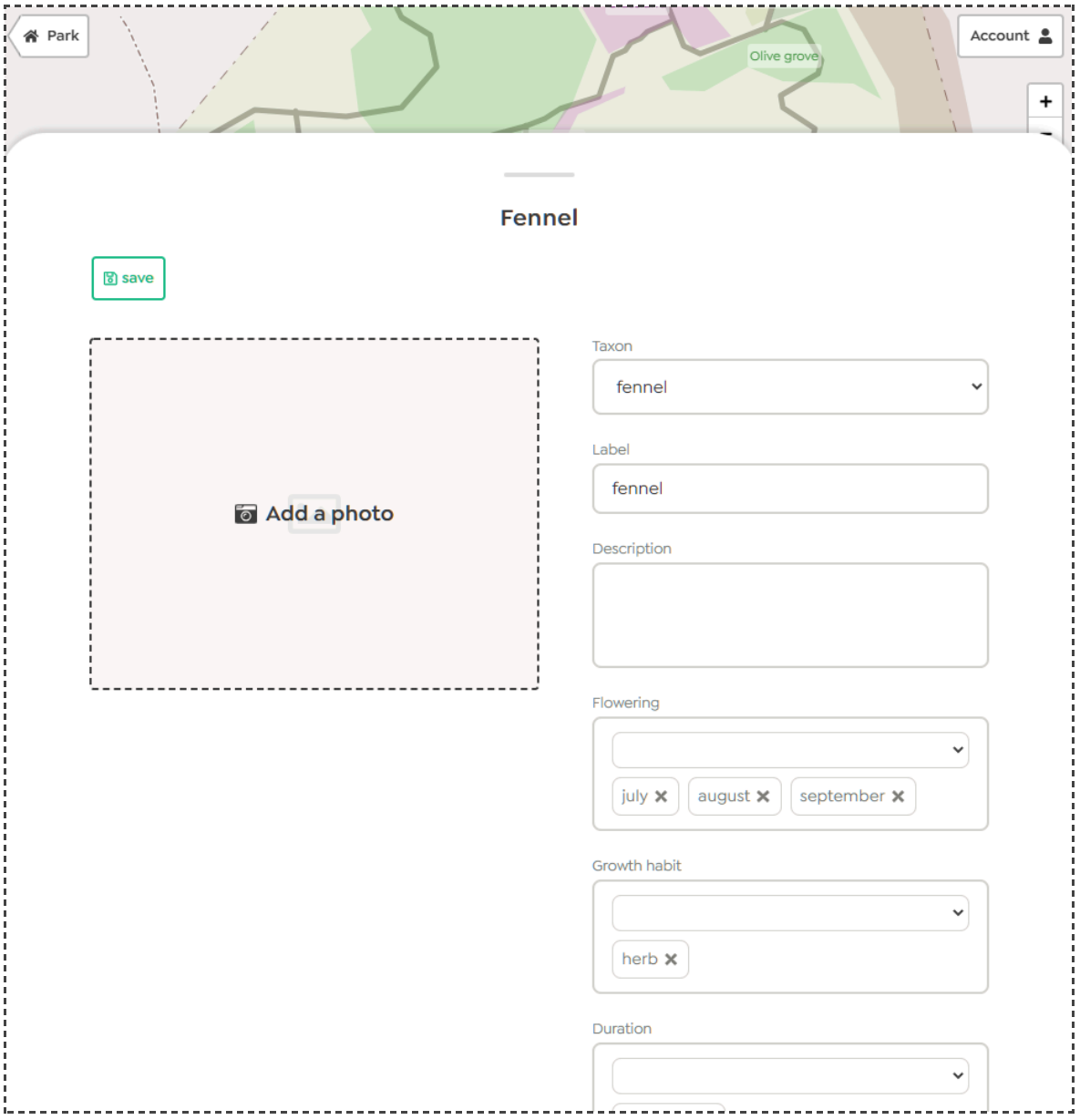


Figure 4.17: Record observation page UI

4.7 Integration

This portion explains how the various components of the web application were combined and made to work together as a cohesive system. Since the architecture includes several different technologies and services, including front-end frameworks, backend servers, real-time communication mechanisms, and databases, it is essential to explain how these disparate parts were integrated to achieve the overall functionality of the app. This section also highlights the challenges faced and the solutions employed to ensure smooth data flow and system performance.

4.7.1 System Overview

The system architecture consists of several core components, all hosted on a Virtual Private Server (VPS) running a Linux operating system. At the forefront is the App Server, built with React and Next.js, which is responsible for rendering the user interface and ensuring a seamless user experience. Complementing this is the API/CMS Server, developed using Node.js and Strapi.js, which manages and delivers content, as well as handles user-related data.

A dedicated Location Server, also based on Node.js, facilitates real-time collaboration and user positioning through the use of Socket.IO and Redis Pub/Sub mechanisms. MariaDB functions as the system's primary relational database, storing all structured backend data securely and efficiently. Lastly, Redis plays a crucial role in real-time data synchronization, particularly for features involving user location sharing and team-based collaboration.

The integration between these components was made possible through several communication mechanisms, including REST and GraphQL APIs, WebSockets, and Redis Pub/Sub for real-time functionality. Below, we explain in detail how these components were integrated and how data flows through the system.

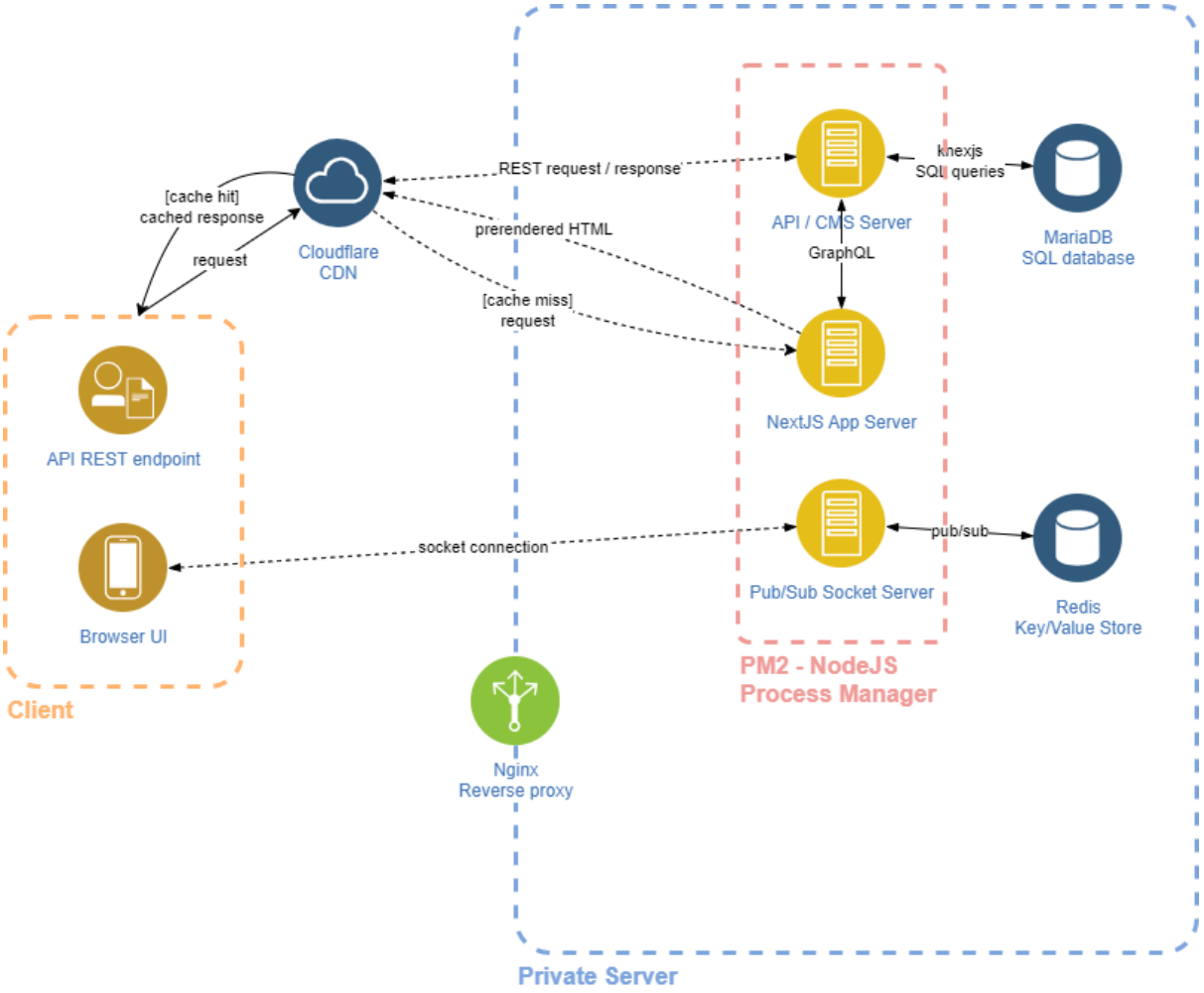


Figure 4.18: Bird's eye view of the system architecture

4.7.2 Frontend and Backend Communication

Next.js, a React-based framework, was chosen for its powerful capability to render pages on both the client and server side. This dual rendering capability supports better performance and search engine optimization. The Next.js server, also referred to as the App Server, works closely with the Strapi.js backend, known as the API/CMS Server, to fetch and dynamically render data on the client's browser.

In terms of data fetching, the App Server communicates with the Strapi.js API to request content and data stored in the MariaDB database. This communication primarily occurs via the GraphQL API exposed by Strapi. For example, when a user navigates to a page detailing a specific park feature, Next.js sends a GraphQL query to Strapi, which then retrieves the relevant data from MariaDB. The data is returned to Next.js, which uses it to render the page content dynamically on the user's browser.

Next.js also employs Server-Side Rendering (SSR) to enhance the application's performance. SSR allows Next.js to pre-generate HTML pages on the server and deliver them to the browser in an optimized format. This is especially beneficial for users on slower connections, as it reduces the time required to load and display content.

To retrieve data from the backend, Next.js issues either GraphQL queries or REST API requests to Strapi.js. Strapi then processes these requests by querying the MariaDB database and returning the necessary information, such as details about park trails, user profiles, or user-submitted observations.

4.7.3 Real-Time Collaboration with Redis Pub/Sub and Socket.IO

The web application includes real-time location-sharing and collaboration features, enabled by the integration of Socket.IO for WebSockets and Redis Pub/Sub for real-time data distribution.

Real-Time Communication

The Location Server, developed using Node.js, manages real-time connections between the server and client browsers. Its primary function is to maintain live updates of user locations within the park and support interactive, team-based features. This server ensures that users can collaborate and track each other’s positions seamlessly as they move throughout the environment.

WebSocket communication is employed to enable these real-time interactions. When a user logs into the application, a WebSocket connection is established between their browser and the Location Server. This persistent connection allows for continuous data exchange without the need for repeated HTTP requests or page reloads, significantly improving the responsiveness and interactivity of the application.

To distribute real-time updates efficiently, the system integrates Redis using its Publish/Subscribe (Pub/Sub) mechanism. This setup works in two key stages. First, as a user moves within the park, their updated location data is sent to the Location Server, which then publishes this information to a Redis channel. Second, all clients that are subscribed to this channel receive the broadcasted updates in real time. This ensures that every connected user has immediate access to the latest location data of their team members, enabling smooth and synchronized collaboration within the application.

Redis Pub/Sub Workflow

The Redis Pub/Sub workflow plays a central role in managing real-time location updates across the system. Redis is configured with Keyspace Notifications to monitor changes in specific keys related to user locations. When a user's position is updated in the system, Redis triggers an event that notifies all relevant subscribers of the change. This event is

captured by the Socket.IO server, which acts as the intermediary between Redis and the client browsers.

Once the event is received, the Socket.IO server immediately broadcasts the updated location data to all connected clients. This ensures that every user receives synchronized, real-time updates about their team members' movements within the park. The overall design is highly scalable, as Redis's lightweight, in-memory architecture allows it to efficiently handle high volumes of message broadcasting without putting strain on system resources. This ensures that real-time functionality remains reliable and performant even as the number of users increases.

4.7.4 Database and API Integration

The backend architecture incorporates a seamless integration between Strapi.js and MariaDB to manage and store structured data. The Strapi.js API/CMS Server communicates directly with the MariaDB database to handle content such as park features, user profiles, and wildlife observations. Strapi uses Knex.js as its query builder, allowing for the efficient execution of complex SQL queries. All user-generated and administrative data is securely stored in MariaDB tables, which are kept inaccessible from the public internet to maintain security.

The data flow begins when the frontend, built with Next.js, sends a request to the Strapi.js API—such as when a user submits a wildlife observation. Strapi processes the request and interacts with MariaDB to either retrieve or store the necessary data. This interaction is typically handled via the GraphQL API, which enables secure, structured, and efficient data exchange between the frontend and backend components.

In terms of security and data access, both the MariaDB and Redis servers are strictly isolated from direct external access. Only internal backend services—namely, the Strapi API server and the Socket.IO-based Location Server—have permission to interact with these databases. This architecture ensures that sensitive data remains protected and only accessible through controlled, authenticated channels.

4.7.5 Conclusion

The integration of various technologies within this web application was critical to delivering real-time functionality, efficient data management, and a seamless user experience. By leveraging Next.js for server-side rendering, Redis Pub/Sub for real-time updates, and Strapi.js for API management, the system was able to meet its performance, scalability, and usability goals. Integration challenges, such as real-time synchronization and secure communication between components, were effectively resolved through the strategic use of these technologies and communication protocols.

4.8 Challenges and Solutions

Here, we delve into the major challenges encountered during the application's development and deployment, as well as the solutions or workarounds that were implemented.

The primary technical challenge was achieving real-time data synchronization across various components, particularly with real-time location sharing and updates. This was addressed by implementing Redis for its Pub/Sub capabilities, which allowed efficient data synchronization and minimized latency in user location updates.

Another challenge was ensuring the app worked seamlessly across various devices and browsers, especially given that visitors might use both desktop and mobile devices with different operating systems. The mobile-first design approach was prioritized, with responsive design principles ensuring compatibility on a wide range of screen sizes. Extensive cross-browser testing was performed, covering popular browsers like Chrome, Safari, Firefox, and Edge. Next.js provided built-in optimizations for handling diverse devices and environments, simplifying the testing and adaptation process.

4.9 Application Features

4.9.1 Observations Sharing

The observations sharing feature enables users to record and share their wildlife or plant observations within the park. This functionality captures data such as the type of observation, location, time, and any relevant notes or images provided by the user. This data is stored in the database and shared with other users through an interactive map, where they can view a collection of observations made by others.

Designing this feature focused heavily on encouraging user participation. We aimed to create a streamlined and intuitive interface for uploading observations, minimizing input requirements while still capturing essential data. Additionally, to enhance engagement, users can view other participants' observations in real-time, fostering a sense of community and collaboration. Gamified elements, such as awarding badges for multiple observations, further encourage user engagement and contributions to the park's data.

4.9.2 Trip Planning

The trip planning functionality within the app allows users to plan their park visits more effectively. Users can set up customized routes, save favorite spots, and even share their itineraries with friends or family members within the application. This feature utilizes

geospatial data to help users explore different areas of the park, offering route suggestions and highlighting points of interest based on their chosen path.

The trip planning tool is designed to be user-friendly, featuring a drag-and-drop interface where users can build their routes. It integrates with the app's map and navigation tools, giving users a preview of distances and nearby amenities. Additionally, users can save and modify their plans, enabling them to revisit or adjust routes based on updated conditions or preferences. This functionality enhances the visitor experience by providing a personalized exploration of the park.

4.9.3 Educational Activities

The app incorporates several educational activities aimed at engaging users and deepening their understanding of the park's ecology. These activities include interactive quizzes, informational pop-ups, and mini-games about the park's flora and fauna. Each educational module is designed to be both informative and entertaining, encouraging users to learn about their surroundings as they explore.

These educational features contribute to the app's role as a learning tool, appealing to users of all ages, especially younger visitors. By integrating educational content into the natural exploration experience, the app not only enhances visitor engagement but also fosters environmental awareness and appreciation.

4.9.4 Interactive Map and Navigation

The interactive map and navigation features form the core of the app's user interface, allowing users to locate themselves, identify points of interest, and navigate through the park. The map interface displays various landmarks, trails, facilities, and user-generated observations, providing a comprehensive view of the park's layout.

For navigation, the app uses GPS data to provide users with real-time location updates, helping them orient themselves within the park. The map interface is equipped with user-friendly controls, allowing visitors to zoom, pan, and tap on icons to access additional information. Points of interest, such as rest areas or scenic viewpoints, are marked on the map to facilitate exploration. These navigation features ensure that users can easily find their way through the park while accessing relevant information about their surroundings.

4.9.5 Real-time Location Sharing and Team Collaboration

The real-time location sharing and team collaboration feature allows users to share their live location with friends or family, enhancing group visits and collaborative activities. Using Redis's Pub/Sub capabilities, this feature enables seamless real-time data updates, allowing users within a shared group to see each other's locations on the map instantly.

This functionality supports team-based activities, such as group hiking or scavenger hunts, by providing real-time visibility of each participant's position. In addition to location sharing, the app allows users to send brief status updates or notifications to others within their group, promoting coordination and communication. These collaborative features enhance the social aspect of park visits, enabling visitors to engage in shared activities and stay connected with their group throughout the exploration.

5. Testing and Evaluation

The effectiveness of the web application in enhancing the visitor experience was systematically evaluated through usability testing. This methodology was chosen to assess how well the application meets user needs and expectations in real-world scenarios, providing valuable insights into its usability, functionality, and overall impact.

5.1 Usability Testing Methodology

5.1.1 Testing Objectives

The primary objectives of the usability testing were to assess the overall intuitiveness and ease of use of the application interface. The testing aimed to identify any potential usability issues that might arise across a range of user experience levels, from beginners to more tech-savvy individuals. Particular attention was given to evaluating the effectiveness of key features such as observation sharing, trip planning, and real-time location sharing, as these are central to the application's functionality.

In addition to evaluating feature performance, the testing also focused on gathering qualitative feedback related to user satisfaction and the perceived value of the application. Insights from this feedback were essential in highlighting areas that work well and uncovering those that require refinement. Ultimately, the testing served as a foundation for identifying opportunities to enhance the user experience in future development iterations.

5.1.2 Participant Selection

For the usability testing, two users with different levels of technological familiarity were selected. This deliberate choice allowed for the assessment of the application's accessibility across varying technical competencies:

- Participant 1: Experienced technology user, familiar with various applications and digital interfaces
- Participant 2: Basic technology user with some smartphone operation knowledge but limited experience with specialized applications

This diversity in user profiles helped evaluate whether the application successfully bridges the gap between different user experience levels.

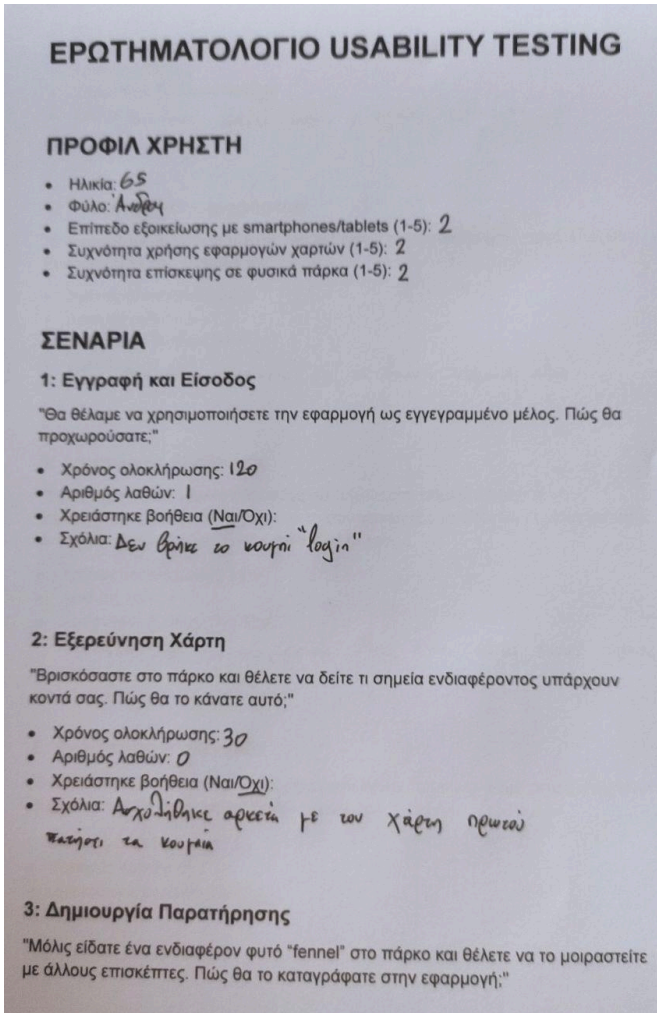
5.1.3 Testing Procedure

The testing process was designed to evaluate user interactions with the application in a natural and insightful way. It began with a pre-test briefing, during which participants were introduced to the purpose and goals of the application. However, they were not given detailed instructions on how to use it, allowing for a more authentic assessment of the interface's intuitiveness.

Participants then engaged in a task-based evaluation, where they were asked to complete a series of activities that reflect the core functionalities of the application. These tasks included creating a user account, finding specific observations on the map, creating and searching for observations, planning a trip within the park, searching for existing trips, using the real-time location sharing feature with group functionality, and participating in educational activities.

An observation method was employed throughout the task execution. Test facilitators closely monitored participants' navigation patterns, the time taken to complete each task, any points of confusion or hesitation, and whether tasks were completed successfully. A think-aloud protocol was also encouraged, where participants verbalized their thoughts while using the app, offering real-time insights into their decision-making process.

Finally, a post-test interview was conducted to gather in-depth, qualitative feedback. Participants were asked about their overall satisfaction, preferences for specific features, any difficulties they encountered, and suggestions for improving the application. This feedback was crucial for identifying usability issues and informing future design and development efforts.



ΕΡΩΤΗΜΑΤΟΛΟΓΙΟ USABILITY TESTING

ΠΡΟΦΙΛ ΧΡΗΣΤΗ

- Ηλικία: 65
- Φύλο: Άνδρας
- Επίπεδο εξοικείωσης με smartphones/tablets (1-5): 2
- Συχνότητα χρήσης εφαρμογών χαρτών (1-5): 2
- Συχνότητα επίσκεψης σε φυσικά πάρκα (1-5): 2

ΣΕΝΑΡΙΑ

1: Εγγραφή και Είσοδος

"Θα θέλαμε να χρησιμοποιήσετε την εφαρμογή ως εγγεγραμμένο μέλος. Πώς θα προχωρούσατε;"

- Χρόνος ολοκλήρωσης: 120
- Αριθμός λαθών: 1
- Χρειάστηκε βοήθεια (Ναι/Όχι):
- Σχόλια: Δεν βρήκε το κουμπί "login"

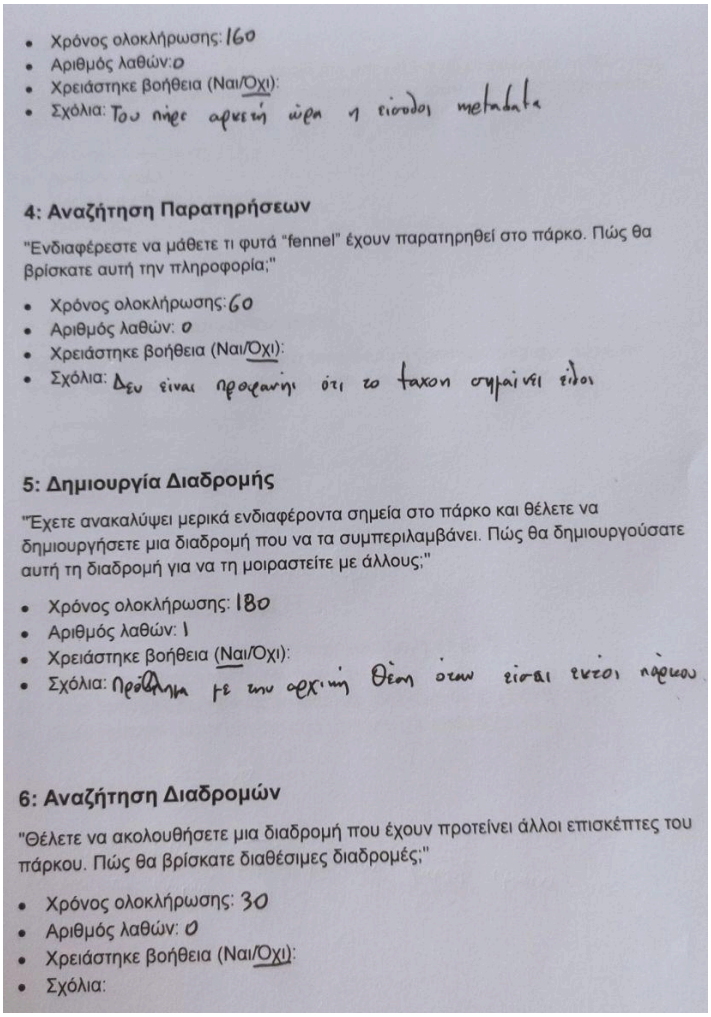
2: Εξερεύνηση Χάρτη

"Βρισκόσαστε στο πάρκο και θέλετε να δείτε τι σημεία ενδιαφέροντος υπάρχουν κοντά σας. Πώς θα το κάνατε αυτό;"

- Χρόνος ολοκλήρωσης: 30
- Αριθμός λαθών: 0
- Χρειάστηκε βοήθεια (Ναι/Όχι):
- Σχόλια: Ασχολήθηκε αρκετά με τον χάρτη πρωτού πατήσει τα κουμπία

3: Δημιουργία Παρατήρησης

"Μόλις είδατε ένα ενδιαφέρον φυτό "fennel" στο πάρκο και θέλετε να το μοιραστείτε με άλλους επισκέπτες. Πώς θα το καταγράφατε στην εφαρμογή;"



- Χρόνος ολοκλήρωσης: 160
- Αριθμός λαθών: 0
- Χρειάστηκε βοήθεια (Ναι/Όχι):
- Σχόλια: Του πήρε αρκετή ώρα η είσοδος metabata

4: Αναζήτηση Παρατηρήσεων

"Ενδιαφέρεστε να μάθετε τι φυτά "fennel" έχουν παρατηρηθεί στο πάρκο. Πώς θα βρίσκατε αυτή την πληροφορία;"

- Χρόνος ολοκλήρωσης: 60
- Αριθμός λαθών: 0
- Χρειάστηκε βοήθεια (Ναι/Όχι):
- Σχόλια: Δεν είναι προφανής ότι το taxon σημαίνει είδος

5: Δημιουργία Διαδρομής

"Έχετε ανακαλύψει μερικά ενδιαφέροντα σημεία στο πάρκο και θέλετε να δημιουργήσετε μια διαδρομή που να τα συμπεριλαμβάνει. Πώς θα δημιουργούσατε αυτή τη διαδρομή για να τη μοιραστείτε με άλλους;"

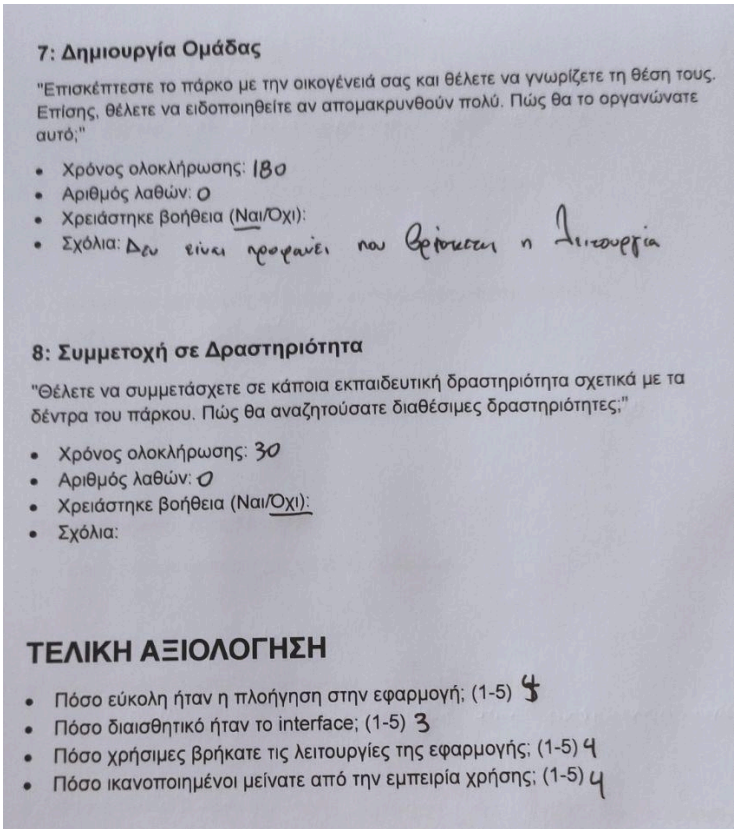
- Χρόνος ολοκλήρωσης: 180
- Αριθμός λαθών: 1
- Χρειάστηκε βοήθεια (Ναι/Όχι):
- Σχόλια: Πρόβλημα με την αρχική θέση όταν είσαι εκτός πάρκου

6: Αναζήτηση Διαδρομών

"Θέλετε να ακολουθήσετε μια διαδρομή που έχουν προτείνει άλλοι επισκέπτες του πάρκου. Πώς θα βρίσκατε διαθέσιμες διαδρομές;"

- Χρόνος ολοκλήρωσης: 30
- Αριθμός λαθών: 0
- Χρειάστηκε βοήθεια (Ναι/Όχι):
- Σχόλια:

Figure 5.1 (left): Usability testing questionnaire (in Greek), page 1, user profile and questions
Figure 5.2 (right): Usability testing questionnaire (in Greek), page 2, questions cont.



7: Δημιουργία Ομάδας

"Επισκέπτεστε το πάρκο με την οικογένειά σας και θέλετε να γνωρίζετε τη θέση τους. Επίσης, θέλετε να ειδοποιηθείτε αν απομακρυνθούν πολύ. Πώς θα το οργανώνατε αυτό;"

- Χρόνος ολοκλήρωσης: 180
- Αριθμός λαθών: 0
- Χρειάστηκε βοήθεια (Ναι/Όχι):
- Σχόλια: Δεν είναι προφανές που βρίσκεται η λειτουργία

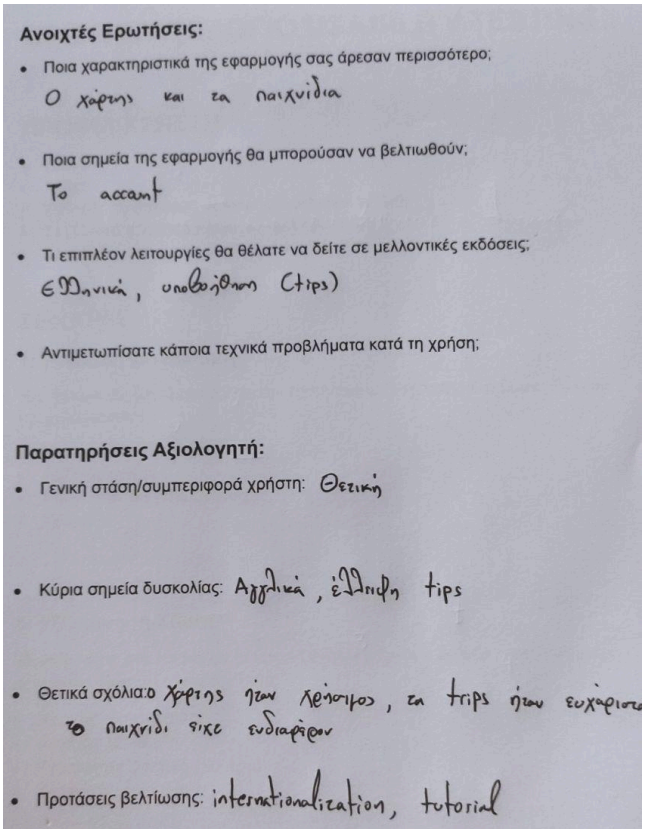
8: Συμμετοχή σε Δραστηριότητα

"Θέλετε να συμμετάσχετε σε κάποια εκπαιδευτική δραστηριότητα σχετικά με τα δέντρα του πάρκου. Πώς θα αναζητούσατε διαθέσιμες δραστηριότητες;"

- Χρόνος ολοκλήρωσης: 30
- Αριθμός λαθών: 0
- Χρειάστηκε βοήθεια (Ναι/Όχι):
- Σχόλια:

ΤΕΛΙΚΗ ΑΞΙΟΛΟΓΗΣΗ

- Πόσο εύκολη ήταν η πλοήγηση στην εφαρμογή; (1-5) 4
- Πόσο διαισθητικό ήταν το interface; (1-5) 3
- Πόσο χρήσιμες βρήκατε τις λειτουργίες της εφαρμογής; (1-5) 4
- Πόσο ικανοποιημένοι μείνατε από την εμπειρία χρήσης; (1-5) 4



Ανοιχτές Ερωτήσεις:

- Ποια χαρακτηριστικά της εφαρμογής σας άρεσαν περισσότερο;
  Ο χάρτης και τα παιχνίδια
- Ποια σημεία της εφαρμογής θα μπορούσαν να βελτιωθούν;
  Το account
- Τι επιπλέον λειτουργίες θα θέλατε να δείτε σε μελλοντικές εκδόσεις;
  Εθνικά, υποβοήθηση (tips)
- Αντιμετωπίσατε κάποια τεχνικά προβλήματα κατά τη χρήση;

Παρατηρήσεις Αξιολογητή:

- Γενική στάση/συμπεριφορά χρήστη: Θετική
- Κύρια σημεία δυσκολίας: Αγγλικά, έλλειψη tips
- Θετικά σχόλια: ο χάρτης ήταν χρήσιμος, τα tips ήταν ευχάριστα το παιχνίδι είχε ενδιαφέρον
- Προτάσεις βελτίωσης: internationalization, tutorial

Figure 5.3 (left): Usability testing questionnaire (in Greek), page 3, questions cont. and final evaluation
Figure 5.4 (right): Usability testing questionnaire (in Greek), page 4, final evaluation cont.

5.2 Testing Results

5.2.1 Task Performance

The performance of both participants varied according to their technical experience:

- Experienced user (Participant 1): Adapted quickly to the interface, completing all tasks efficiently with minimal hesitation. This participant intuitively understood the application's navigation structure and quickly discovered advanced features without prompting.
- Basic user (Participant 2): Required more time to familiarize themselves with the interface but successfully completed most tasks without assistance. Initial navigation presented some challenges, but the user demonstrated increasing comfort with the application throughout the session.

This performance differential was expected but importantly revealed that the application's core functionality remained accessible even to less technically experienced users.

5.2.2 Feature Reception

Several features of the application received particularly positive feedback from participants during usability testing. The interactive maps stood out as a favorite, with users appreciating the clear visual representation of the park and the intuitive navigation through spatial information. This feature made exploring the park more engaging and accessible.

The trip planning functionality was also highly valued. Participants highlighted the usefulness of creating custom routes and sharing them with others, noting that it added a layer of personalization and collaborative planning to their park experience. Similarly, the real-time location sharing feature was praised for its potential to improve safety and coordination, especially during group visits or educational excursions.

Lastly, the observation system—which allows users to record and share wildlife sightings—was found to be both engaging and educational. Participants enjoyed the interactive aspect of contributing their own observations, as well as learning from others, which enriched their overall experience with the application.

5.2.3 Identified Usability Issues

The usability testing also revealed several areas for improvement. One of the key issues identified was navigation clarity. The basic user occasionally had difficulty locating specific features within the application, indicating a need for more prominent or intuitive navigation cues to guide users through key functionalities.

Another point of feedback was the lack of clear feedback mechanisms. Both participants expressed a desire for more explicit confirmation when actions were successfully completed—for example, submitting an observation or saving a trip plan. Implementing visual or auditory cues could enhance user confidence and satisfaction.

The first-time user experience was also noted as an area needing attention. The basic user would have benefited from a brief onboarding tutorial or guided walkthrough upon initial use, to help them understand the layout and capabilities of the app more quickly.

Lastly, there was some uncertainty regarding connection status handling. Both users were unsure how the application would respond in situations with intermittent or lost connectivity. Providing clearer indicators and fallback behavior for offline scenarios would help set user expectations and improve reliability.

5.3 User Feedback and Improvements

5.3.1 Implemented Improvements

Following the usability testing, several immediate enhancements were made to address the feedback received from participants. Enhanced visual cues were introduced, making key navigation elements more prominent and easier to locate, which improved overall discoverability. To address the lack of user feedback, confirmation messages were added to acknowledge successful actions such as creating observations or saving trips.

Additionally, the authentication process—including account creation and login—was streamlined to reduce friction points that had been identified during testing, resulting in a smoother user onboarding experience.

5.3.2 Future Enhancement Priorities

Beyond the initial updates, several future development priorities have been identified to further improve the application. A key focus will be implementing introductory tutorials and contextual walkthroughs to support first-time users and enhance their understanding of the platform. Internationalization is also planned, enabling support for multiple languages to make the application accessible to a broader user base.

Another priority is the addition of real-time communication features, such as direct messaging between group members, to complement the location sharing functionality. To address concerns about inconsistent connectivity in remote areas, offline functionality will be developed, allowing users to continue using key features without a stable internet connection. Finally, ongoing work will focus on accessibility enhancements, ensuring the application meets the diverse needs of users with varying abilities.

5.3.3 Evaluation Conclusions

The usability testing confirmed that the application successfully achieves its primary objective of enhancing the natural park experience for visitors with varying levels of technical proficiency. The intuitive interface design allows even users with basic technological familiarity to access and benefit from the application's core features.
The interactive map functionality, observation sharing, and real-time location features proved particularly effective in meeting user needs and expectations. While areas for improvement were identified, the fundamental architecture and approach of the application were validated through this evaluation process.
This testing phase provided valuable insights that will guide future development, ensuring that the application continues to evolve in alignment with user needs and preferences. The methodology employed -observing real users attempting real tasks- proved effective in uncovering both strengths and opportunities for enhancement that might not have been apparent through theoretical analysis alone.

6. Conclusion and Future Work

This chapter summarizes the key findings and contributions of the project and discusses potential future work and enhancements to the application.

6.1 Summary of the thesis

This thesis presented the design and implementation of a web-based platform aimed at enhancing visitor experiences at the Technical University of Crete Natural Park. The platform successfully integrates several innovative features that contribute to both the educational and social aspects of park visits:

- **Real-Time Location Sharing System**: Leveraging Redis Pub/Sub and WebSocket communications, this feature enables safe group exploration and collaborative activities within the park. It allows visitors to form groups, share their locations in real-time, and receive alerts when group members stray too far, thereby addressing safety concerns while fostering a sense of community among park visitors.

- **Comprehensive Observation System**: The platform allows visitors to document and share their wildlife and plant sightings, thereby contributing to citizen science efforts. This system creates a growing repository of observations, complete with taxonomic classifications, images, and location data. It serves both educational and scientific purposes, enabling users to contribute valuable data while learning more about the park's ecosystems.

- **Educational Framework with Traditional Ecological Knowledge**: The platform incorporates interactive activities and content about traditional ecological knowledge. Through gamified elements and informative content, visitors are able to engage with the park's biodiversity, conservation principles, and the cultural significance of local plants and animals, all within an interactive, engaging format.

- **Flexible Trip Planning System**: This feature supports both individual and group park exploration. Users can create, save, and share custom routes with waypoints, observations, and commentaries, enriching the experience for both the user and others who follow their planned routes.

The implementation of the platform uses modern web technologies, including TypeScript, React, Next.js, and Node.js, ensuring scalability and performance through careful architectural design. The system demonstrates how complex, real-time web applications can effectively meet educational and scientific goals, providing an innovative tool for enhancing natural park visits.

6.2 Contribution to the field

This work makes several contributions to the field of environmental education and park management technology:

6.2.1 Technical Contributions

This work offers significant contributions to the domains of environmental education, citizen science, and park management technologies. From a **technical standpoint**, it demonstrates the successful integration of real-time features into a natural park application. The use of **Redis Pub/Sub mechanisms** and **WebSocket communication** has proven effective in delivering responsive, real-time location-sharing capabilities, even in the challenging conditions typical of outdoor environments. This implementation showcases how modern technologies can maintain performance and reliability in remote and distributed settings.

Another key technical innovation is the **efficient management of taxonomic data** in conjunction with **traditional ecological knowledge**. The underlying database design provides a flexible yet structured model that organizes complex scientific classifications alongside user-generated content and culturally significant knowledge, promoting both scientific rigor and cultural relevance.

6.2.2 Educational Contributions

In terms of **educational contributions**, the platform effectively bridges the gap between scientific documentation and public engagement. It translates complex ecological data into interactive and accessible formats suitable for a wide audience, from casual visitors to educators and students. Furthermore, the application presents a **framework for incorporating traditional ecological knowledge** into digital platforms, preserving and presenting this valuable information in a meaningful, research-friendly format.

The system also serves as a **model for citizen science initiatives** within protected natural areas. Through its intuitive observation system, it empowers non-specialist users to contribute scientifically relevant data, demonstrating how thoughtful interface design can support rigorous public data collection. This model enhances participatory research while fostering environmental stewardship among park visitors.

6.2.3 Research Applications

From a **research perspective**, the platform facilitates the systematic collection of structured citizen science data. With timestamped and geotagged observations, it builds a growing dataset that supports **long-term environmental monitoring** and enables ecological trend analysis over time. Additionally, by digitizing traditional ecological knowledge, the platform contributes to the **preservation and scholarly study of cultural heritage** tied to natural environments.

6.3 Limitations and future work

6.3.1 Current Limitations

Despite its strengths, the application has several current **limitations**. Technically, one of the most significant is the **lack of offline functionality**. As most features depend on a stable internet connection, users in remote park areas may face difficulties accessing content or recording observations. Another technical constraint is the need for **manual moderation of user-generated content**, which could become burdensome as the platform scales and more observations and comments are submitted.

From a content perspective, the platform currently offers **limited language support**, which restricts accessibility for non-native speakers and international visitors. Furthermore, the **manual process of adding new taxonomic entries** through the CMS poses a bottleneck, as it requires administrative oversight without a streamlined contribution workflow for experts. In terms of engagement, **gamification elements** remain basic, lacking features like achievements, progress tracking, or user challenges that could further motivate participation, particularly among younger audiences.

6.3.2 Future Work

To address these issues, several avenues of **future work** have been identified. On the technical front, implementing **Progressive Web App (PWA)** features would introduce **offline capabilities**, enabling users to download essential park information, record observations offline, and sync data once reconnected. Additionally, **automated content moderation** using machine learning could assist in flagging potentially inaccurate or inappropriate content, significantly improving scalability. Enhancing **spatial query functionality** would further support advanced visualization and ecological analysis.

Several **feature additions** are also planned. These include **Augmented Reality (AR)** support for real-time species identification, an **achievement-based gamification system**, and **interactive virtual tours** that expand the application's educational and outreach potential. Integrating **real-time environmental monitoring** via IoT sensors could enrich the

user experience and contribute to ongoing scientific research. Advanced research tools are also proposed to allow deeper analysis of collected data by ecologists and other specialists.

In terms of **content expansion**, the introduction of **multi-language support** will enhance accessibility, especially in multilingual or international contexts. Additional **educational activities and games** will be developed to engage a broader range of learners. Continued documentation and integration of **traditional ecological knowledge** will further preserve and disseminate cultural insights tied to local ecosystems.

Finally, the platform presents strong opportunities for **integration with external systems**. Connections to global biodiversity databases such as **GBIF** could position local observations within a broader scientific context. Integration with **environmental sensor networks** would provide dynamic environmental data, supporting both visitor experiences and academic research. Additionally, developing standardized **data export features** would support external research efforts, making citizen science contributions more impactful within the scientific community.

6.4 Final thoughts and recommendations

The successful implementation of this platform highlights the significant role technology can play in enhancing both visitor experiences and environmental education in natural park settings. Based on the insights gained during development and usability testing, we offer several recommendations for future implementations and improvements:

6.4.1 For Implementation

First and foremost, **prioritizing user experience and accessibility** is crucial. The platform should cater to users of varying technical abilities, as demonstrated by our usability testing with participants of different experience levels. By focusing on intuitive interfaces, the application can reach a wider audience, maximizing its educational and social impact.

Additionally, it's important to **plan for offline capabilities from the outset**. Given the unpredictable connectivity in many natural environments, designing the platform with offline functionality in mind will prevent costly retrofitting later on and ensure a better user experience, even in remote areas with limited internet access.

Another key consideration is the need to **implement robust data validation**. This is essential to ensure that user-generated content maintains high scientific integrity while still being simple and intuitive for users to submit. A comprehensive validation mechanism will support the reliability of the data without hindering user participation.

Lastly, it's important to **design for scalability and maintainability**. Building modular and well-documented code and architecture will allow the platform to evolve as the user base

grows and requirements change. This approach ensures the long-term success and adaptability of the application.

6.4.2 For Content Management

For effective **content management**, we recommend developing **comprehensive moderation procedures**. Efficient workflows for reviewing and managing user submissions will help maintain content quality while encouraging ongoing user engagement. Balancing quality control with timely content approval is critical for keeping the community active.

It is also important to **plan for multi-language support** early in the design process. While initial implementation may focus on a single language, structuring content for easy expansion to additional languages will facilitate smoother internationalization and broaden the application's reach.

Creating clear **structured processes for adding scientific content** is vital for ensuring taxonomic information remains accurate and up to date. Establishing protocols for the creation and updating of scientific data will streamline the process while preserving scientific integrity.

Furthermore, **documenting traditional knowledge systematically** should be a priority. Developing respectful methodologies for gathering, validating, and integrating traditional ecological knowledge will honor cultural ownership while making this valuable information accessible for educational use.

6.4.3 For Future Development

In terms of **future development**, investigating **Augmented Reality (AR) and mixed reality opportunities** could lead to more immersive, interactive experiences, particularly for species identification and ecological visualization. These technologies have the potential to significantly enhance visitor engagement by offering dynamic and informative ways to explore the park.

Expanding **educational game features** is also a key recommendation. Developing more sophisticated and engaging interactive experiences will appeal to diverse learning styles and age groups, deepening the educational impact of park visits and making learning more enjoyable.

Finally, **enhancing research data collection capabilities** will be critical. Refining observation tools to capture more detailed and structured scientific data will increase the platform's scientific value. These improvements should be made without compromising the simplicity and ease of use for the average user, thus maximizing the contributions from citizen scientists while maintaining an accessible experience.

7. References

This section contains a comprehensive list of cited literature & research used in the thesis:

1. Bodzin, A. M., Anastasio, D., & Kulo, V. (2014). Designing Google Earth activities for learning Earth and environmental science. Teaching Science and Investigating Environmental Issues with Geospatial Technology, 213-232 DOI

2. Goodchild, M. F. (2007). Citizens as sensors: The world of volunteered geography. GeoJournal 69, 211-221 DOI

3. Brown, G., & Weber, D. (2011). Public participation GIS: A new method for national park planning. Landscape and Urban Planning 102, 1-15 DOI

4. Haklay, M. (2013). Citizen Science and Volunteered Geographic Information: Overview and typology of participation. Crowdsourcing Geographic Knowledge: Volunteered Geographic Information (VGI) in Theory and Practice, 105-122. Springer, Dordrecht DOI

5. Steve Kelling, Daniel Fink, Frank A. La Sorte, Alison Johnston, Nicholas E. Bruns, Wesley M. Hochachka (2015). Taking a 'Big Data' approach to data quality in a citizen science project. Ambio 44(Suppl. 4), 601-611 DOI

6. Elwood, S., Goodchild, M.F., & Sui, D.Z. (2012). Researching Volunteered Geographic Information: Spatial Data, Geographic Research, and New Social Practice. Annals of the Association of American Geographers, 102(X) 2012, pp. 1–20 DOI

7. Sullivan, B. L., Aycrigg, J. L., Barry, J. H., Bonney, R. E., Bruns, N., Cooper, C. B., ... & Kelling, S. (2014). The eBird enterprise: An integrated approach to development and application of citizen science. Biological Conservation, 169, 31-40 DOI

8. Silvertown, J. (2009). A new dawn for citizen science. Trends in ecology & evolution, 24(9), 467-471 DOI

9. Deterding, S., Dixon, D., Khaled, R., & Nacke, L. (2011). From game design elements to gamefulness: defining gamification. Proceedings of the 15th international academic MindTrek conference: Envisioning future media environments, 9-15 DOI

10. O'Hara, K. (2008). Understanding geocaching practices and motivations. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 1177-1186 DOI

11. Hanus, M. D., & Fox, J. (2015). Assessing the effects of gamification in the classroom: A longitudinal study on intrinsic motivation, social comparison, satisfaction, effort, and academic performance. Computers & Education, 80, 152-161 DOI

12. Snyder, C. (2003). Paper Prototyping: The Fast and Easy Way to Design and Refine User Interfaces. Morgan Kaufmann link

13. Cooper, A., Reimann, R., & Cronin, D. (2007). About Face 3: The Essentials of Interaction Design. Wiley. ISBN: 0470084111 link

14. Sanders, E. B. N., & Stappers, P. J. (2008). Co-creation and the new landscapes of design. Co-design, 4(1), 5-18 DOI

8. REST web services

The web application was designed with a RESTful (Representational State Transfer) architecture. This is an architectural style that structures an application around resources, and uses HTTP protocols as its foundation. Each URL is considered a resource and is accessed using standard HTTP methods, such as GET, POST, PUT, and DELETE.
RESTful APIs are stateless, meaning that each HTTP request happens independently. Without state being stored on the server, applications remain lightweight and simple. The REST API of Strapi.js, the CMS and backend of our application, was crucial for the communication between the frontend and backend.
Creating, reading, updating, or deleting content all went through the API, with data sent and received in JSON format. This setup provided a seamless and efficient way for the frontend of the application to communicate with the backend, allowing for dynamic and interactive user interfaces.
The choice of RESTful architecture made it easier to scale the application, improve performance, and support the real-time data needs of the features in the application such as real-time location sharing and interactive map navigation.

We list the Representational state transfer (REST) web services provided by the system. The data format supported is JSON.
The base URL is the same as the CMS app's URL

8.1 Register user

URL	`{base URL}/api/auth/local/register`
Method	`POST`
Description	`Creates a new user and returns it, along with a jwt web token to identify the user in future requests`
Body: `JSON (type shown in typescript interface notation)`	

`{`
`  username: string; //the username of the user`
`  email: string; //the email of the user`
`  password: string; //the password of the user`
`}`	

Example: `/api/auth/local/register (POST)`	

```
{
    "jwt": "xxx",
    "user": {
        "id": 10,
        "username": "test",
        "email": "test@test.test",
        "provider": "local",
        "confirmed": true,
        "blocked": false,
        "team": null,
        "createdAt": "2023-03-02T18:05:00.943Z",
        "updatedAt": "2023-03-02T18:05:00.943Z",
        "team_radius": null
    }
}
```


8.2 Login user

URL	{base URL}/api/auth/local
Method	POST
Description	Login an existing user and return it, along with a jwt web token to identify the user in future requests

Body: JSON (type shown in typescript interface notation)

```
{
  identifier: string; //the username or email of the user
  password: string; //the password of the user
}
```

Example: /api/auth/local (POST)


```
{
    "jwt": "xxx",
    "user": {
        "id": 10,
        "username": "test",
        "email": "test@test.test",
        "provider": "local",
        "confirmed": true,
        "blocked": false,
        "team": null,
        "createdAt": "2023-03-02T18:05:00.943Z",
        "updatedAt": "2023-03-02T18:05:00.943Z",
        "team_radius": null
    }
}
```

8.3 Get all observations

URL	{base URL}/api/observations?populate={object} &sort={sortBy} &pagination[withCount]={withCount} &pagination[page]={page}&pagination[pageSize]={pageSize} &pagination[start]={pageStart}&pagination[limit]={pageLimit}
Method	GET
Description	Returns the observation list
Parameters	object: the name of the nested object(s) to populate in the response. Can use dot notation to further expand nested objects. Can use a wildcard (*) to populate all objects. sortBy: the field to sort the list by page: which page to display (when paginating by page) pageSize: the size of the page to display (when paginating by page) pageStart: position of first list entry to return (when paginating by offset) pageLimit: number of list entries to return (when paginating by offset) withCount: boolean to toggle displaying total number of entries (default: true)

Example: /api/observations (GET)


```
{
    "data": [
        {
            "id": 19,
            "attributes": {
                "label": "a fennel",
                "description": "found here",
                "createdAt": "2022-05-13T11:17:05.193Z",
                "updatedAt": "2022-05-23T08:54:05.178Z"
            }
        },
    ],
    "meta": {
        "pagination": {
            "page": 1,
            "pageSize": 25,
            "pageCount": 1,
            "total": 1
        }
    }
}
```


8.4 Get an observation

URL	{base URL}/api/observations/{id}?populate={object}
Method	GET
Description	Returns the observation with the specified id

Parameters	id: the unique identifier of the observation object: the name of the nested object(s) to populate in the response. Can use dot notation to further expand nested objects. Can use a wildcard (*) to populate all objects.

Example: /api/observations/19?&populate=position&populate=meta_values.metadatum (GET)


```
{
    "data": {
        "id": 19,
        "attributes": {
            "label": "a fennel",
            "description": "found here",
            "createdAt": "2022-05-13T11:17:05.193Z",
            "updatedAt": "2022-05-23T08:54:05.178Z",
            "meta_values": [
                {
                    "id": 89,
                    "value": "Perennial",
                    "metadatum": {
                        "data": {
                            "id": 10,
                            "attributes": {
                                "label": "duration",
                                "datatype": "list",
                                "description": null,
                                "createdAt": "2022-05-12T06:28:27.531Z",
                                "updatedAt": "2022-05-23T08:45:12.460Z"
                            }
                        }
                    }
                },
                {
                    "id": 90,
                    "value": "July",
                    "metadatum": {
                        "data": {
                            "id": 11,
                            "attributes": {
                                "label": "flowering",
                                "datatype": "list",
                                "description": null,
                                "createdAt": "2022-05-12T06:29:04.488Z",
                                "updatedAt": "2022-05-12T06:55:38.255Z"
                            }
                        }
                    }
                },
                {
                    "id": 91,
                    "value": "Crete",
                    "metadatum": {
                        "data": {
                            "id": 12,
                            "attributes": {
                                "label": "nativity",
                                "datatype": "list",
                                "description": null,
                                "createdAt": "2022-05-12T06:29:16.681Z",
                                "updatedAt": "2022-05-24T10:45:19.409Z"
```

```
                            }
                        }
                    }
                },
                {
                    "id": 92,
                    "value": "food",
                    "metadatum": {
                        "data": {
                            "id": 14,
                            "attributes": {
                                "label": "uses",
                                "datatype": "list",
                                "description": null,
                                "createdAt": "2022-05-12T06:33:08.552Z",
                                "updatedAt": "2022-05-24T10:46:15.761Z"
                            }
                        }
                    }
                }
            ]
        }
    },
    "meta": {}
}
```


8.5 Create an observation

URL	{base URL}/api/observations/
Method	POST
Description	Create a new observation and return it

Body: JSON (type shown in typescript interface notation)

```
{
  data: {
    label: string;
    description?: string;
    position?: {
      latitude: number;
      longitude: number;
    };
    meta_values?: [
      {
        metadatum: string,
        value: string,
      };
    ];
    image?: number;
    taxon?: number;
    trips?: number[];
    comments?: number[];
    creator?: number;
    fav_users?: number[];
  }
```

}

8.6 Update an observation

URL	{base URL}/api/observations/{id}
Method	PUT
Description	Update an existing observation and return it
Parameters	id: the unique identifier of the observation
Body: JSON (type shown in typescript interface notation)	

```
{
  data: {
    label: string;
    description?: string;
    position?: {
      latitude: number;
      longitude: number;
    };
    meta_values?: [
      {
        metadatum: string;
        value: string;
      };
    ];
    image?: number;
    taxon?: number;
    trips?: number[];
    comments?: number[];
    creator?: number;
    fav_users?: number[];
  }
}
```

8.7 Get all activities

URL	{base URL}/api/activities?populate={object} &sort={sortBy} &pagination[withCount]={withCount} &pagination[page]={page}&pagination[pageSize]={pageSize} &pagination[start]={pageStart}&pagination[limit]={pageLimit}
Method	GET
Description	Returns the activity list
Parameters	object: the name of the nested object(s) to populate in the response. Can use dot notation to further expand nested objects. Can use a wildcard (*) to populate all

	objects. sortBy: the field to sort the list by page: which page to display (when paginating by page) pageSize: the size of the page to display (when paginating by page) pageStart: position of first list entry to return (when paginating by offset) pageLimit: number of list entries to return (when paginating by offset) withCount: boolean to toggle displaying total number of entries (default: true)

Example: /api/activities (GET)


```
{
    "data": [
        {
            "id": 2,
            "attributes": {
                "label": "Heartbeat of a Tree",
                "age_min": 4,
                "age_max": null,
                "players_min": 1,
                "players_max": null,
                "month_min": 2,
                "month_max": 5,
                "description": "A tree is a living creature.\nIt eats, rests, breathes and circulates its \"blood\" much as we do. The heartbeat of a tree is a wonderful crackling, gurgling flow of life. \nThe best time to hear the forest heartbeat is in early spring, when the trees send first surges of sap upwards to their branches preparing them for another season of growth.",
                "createdAt": "2022-05-25T19:05:19.724Z",
                "updatedAt": "2022-05-25T19:06:19.983Z"
            }
        },
        {
            "id": 3,
            "attributes": {
                "label": "Unnature trail",
                "age_min": 5,
                "age_max": 13,
                "players_min": 1,
                "players_max": 30,
                "month_min": null,
                "month_max": null,
                "description": "This game is played to introduce the concepts of protective coloration and adaptation. as well as to enhance children's observational skills. \nA benefit of this increased visual awareness is that children become much more careful about littering outdoors.",
                "createdAt": "2022-05-25T19:32:33.370Z",
                "updatedAt": "2022-05-25T19:34:51.875Z"
            }
        }
    ],
    "meta": {
        "pagination": {
            "page": 1,
            "pageSize": 25,
            "pageCount": 1,
            "total": 2
        }
    }
}
```

8.8 Get an activity

URL	{base URL}/api/activities/{id}?populate={object}
Method	GET
Description	Returns the activity with the specified id
Parameters	id: the unique identifier of the activity object: the name of the nested object(s) to populate in the response. Can use dot notation to further expand nested objects. Can use a wildcard (*) to populate all objects.

Example: /api/activities/2 (GET)


```
{
    "data": {
        "id": 2,
        "attributes": {
            "label": "Heartbeat of a Tree",
            "age_min": 4,
            "age_max": null,
            "players_min": 1,
            "players_max": null,
            "month_min": 2,
            "month_max": 5,
            "description": "A tree is a living creature.\nIt eats, rests, breathes and circulates its \"blood\" much as we do. The heartbeat of a tree is a wonderful crackling, gurgling flow of life. \nThe best time to hear the forest heartbeat is in early spring, when the trees send first surges of sap upwards to their branches preparing them for another season of growth.",
            "createdAt": "2022-05-25T19:05:19.724Z",
            "updatedAt": "2022-05-25T19:06:19.983Z"
        }
    },
    "meta": {}
}
```


8.9 Create a comment

URL	{base URL}/api/comments/
Method	POST
Description	Create a new comment and return it

Body: JSON (type shown in typescript interface notation)

```
{
  data: {
    position?: {
      latitude: number;
      longitude: number;
    };
    text: string;
```

```
    activity?: number;
    observation?: number;
    trip?: number;
    place?: number;
    creator?: number;
  }
}
```

8.10 Get all comments

URL	{base URL}/api/comments?populate={object} &sort={sortBy} &pagination[withCount]={withCount} &pagination[page]={page}&pagination[pageSize]={pageSize} &pagination[start]={pageStart}&pagination[limit]={pageLimit}
Method	GET
Description	Return all comments
Parameters	object: the name of the nested object(s) to populate in the response. Can use dot notation to further expand nested objects. Can use a wildcard (*) to populate all objects. sortBy: the field to sort the list by page: which page to display (when paginating by page) pageSize: the size of the page to display (when paginating by page) pageStart: position of first list entry to return (when paginating by offset) pageLimit: number of list entries to return (when paginating by offset) withCount: boolean to toggle displaying total number of entries (default: true)

Example: /api/comments/ (GET)

```
{
    "data": [
        {
            "id": 3,
            "attributes": {
                "text": "test comment",
                "createdAt": "2023-03-04T09:51:33.503Z",
                "updatedAt": "2023-03-04T09:51:33.503Z"
            }
        }
    ],
    "meta": {
        "pagination": {
            "page": 1,
            "pageSize": 25,
            "pageCount": 1,
            "total": 1
        }
    }
}
```

8.11 Get a comment

URL	{base URL}/api/comments/{id}?populate={object}
Method	GET
Description	Returns the comment with the specified id
Parameters	id: the unique identifier of the comment object: the name of the nested object(s) to populate in the response. Can use dot notation to further expand nested objects. Can use a wildcard (*) to populate all objects.

Example: /api/comments/3?populate=observation (GET)


```
{
    "data": {
        "id": 3,
        "attributes": {
            "text": "test comment",
            "createdAt": "2023-03-04T09:51:33.503Z",
            "updatedAt": "2023-03-04T09:51:33.503Z",
            "observation": {
                "data": {
                    "id": 21,
                    "attributes": {
                        "label": "Saw a weasel",
                        "description": "It was very cute",
                        "createdAt": "2022-05-24T21:31:51.030Z",
                        "updatedAt": "2022-05-24T21:31:51.030Z"
                    }
                }
            }
        }
    },
    "meta": {}
}
```


8.12 Get all metadata types

URL	{base URL}/api/metadata?populate={object} &sort={sortBy} &pagination[withCount]={withCount} &pagination[page]={page}&pagination[pageSize]={pageSize} &pagination[start]={pageStart}&pagination[limit]={pageLimit}
Method	GET
Description	Returns all metadata types
Parameters	object: the name of the nested object(s) to populate in the response. Can use dot notation to further expand nested objects. Can use a wildcard (*) to populate all objects. sortBy: the field to sort the list by

	page: which page to display (when paginating by page) pageSize: the size of the page to display (when paginating by page) pageStart: position of first list entry to return (when paginating by offset) pageLimit: number of list entries to return (when paginating by offset) withCount: boolean to toggle displaying total number of entries (default: true)

Example: /api/metadata (GET)


```
{
    "data": [
        {
            "id": 9,
            "attributes": {
                "label": "growth habit",
                "datatype": "list",
                "description": null,
                "createdAt": "2022-05-12T06:26:29.713Z",
                "updatedAt": "2022-05-23T08:44:57.980Z"
            }
        },
        {
            "id": 10,
            "attributes": {
                "label": "duration",
                "datatype": "list",
                "description": null,
                "createdAt": "2022-05-12T06:28:27.531Z",
                "updatedAt": "2022-05-23T08:45:12.460Z"
            }
        },
        {
            "id": 11,
            "attributes": {
                "label": "flowering",
                "datatype": "list",
                "description": null,
                "createdAt": "2022-05-12T06:29:04.488Z",
                "updatedAt": "2022-05-12T06:55:38.255Z"
            }
        },
        {
            "id": 12,
            "attributes": {
                "label": "nativity",
                "datatype": "list",
                "description": null,
                "createdAt": "2022-05-12T06:29:16.681Z",
                "updatedAt": "2022-05-24T10:45:19.409Z"
            }
        },
        {
            "id": 13,
            "attributes": {
                "label": "distribution",
                "datatype": "list",
                "description": null,
                "createdAt": "2022-05-12T06:29:32.380Z",
                "updatedAt": "2022-05-24T21:24:34.423Z"
            }
        },
```

```
        {
            "id": 14,
            "attributes": {
                "label": "uses",
                "datatype": "list",
                "description": null,
                "createdAt": "2022-05-12T06:33:08.552Z",
                "updatedAt": "2022-05-24T10:46:15.761Z"
            }
        },
        {
            "id": 15,
            "attributes": {
                "label": "body length (cm)",
                "datatype": "number",
                "description": null,
                "createdAt": "2022-05-24T21:28:00.696Z",
                "updatedAt": "2022-05-24T21:28:00.696Z"
            }
        }
    ],
    "meta": {
        "pagination": {
            "page": 1,
            "pageSize": 25,
            "pageCount": 1,
            "total": 7
        }
    }
}
```


8.13 Get a metadatum type

URL	{base URL}/api/metadata/{id}?populate={object}
Method	GET
Description	Returns a metadatum with the specified id
Parameters	id: the unique identifier of the metadatum object: the name of the nested object(s) to populate in the response. Can use dot notation to further expand nested objects. Can use a wildcard (*) to populate all objects.

Example: /api/metadata/14?populate=list_values (GET)


```
{
    "data": {
        "id": 14,
        "attributes": {
            "label": "uses",
            "datatype": "list",
            "description": null,
            "createdAt": "2022-05-12T06:33:08.552Z",
            "updatedAt": "2022-05-24T10:46:15.761Z",
            "list_values": [
```

```
                {
                    "id": 63,
                    "value": "food"
                },
                {
                    "id": 64,
                    "value": "materials"
                },
                {
                    "id": 65,
                    "value": "medicine"
                },
                {
                    "id": 66,
                    "value": "environmental"
                }
            ]
        }
    },
    "meta": {}
}
```


8.14 Get the park

URL	{base URL}/api/park?populate={object}
Method	GET
Description	Returns the park
Parameters	object: the name of the nested object(s) to populate in the response. Can use dot notation to further expand nested objects. Can use a wildcard (*) to populate all objects.

Example: /api/park?populate=images (GET)


```
{
    "data": {
        "id": 1,
        "attributes": {
            "label": "TUC Flora & Fauna Preservation Park",
            "description": "At a time that living organisms are disappearing at an alarming rate,
especially through habitat destruction, the Park for the Preservation of Flora and Fauna of the Technical
University of Crete, which extends to 30 hectares, constitutes a small but important area where native
plants and co-existing animals are protected and can develop without human intervention.\n\nPlease do not
bring dogs to the Park. Also bicycles are not allowed in the area.",
            "closed_unplanned": false,
            "open_hours": "mon-fri: 07:00-14:00, 17:00-21:00\nsat-sun: 08:00-12:00, 17:00-21:00",
            "createdAt": "2022-04-10T06:49:52.700Z",
            "updatedAt": "2022-05-11T13:30:32.045Z",
            "email": "park@mail.tuc.gr",
            "phone": "+302821037071",
            "images": {
                "data": [
                    {
                        "id": 44,
```

```
                        "attributes": {
                            "name": "logo_park_el (1).png",
                            "alternativeText": "logo_park_el (1).png",
                            "caption": "logo_park_el (1).png",
                            "width": 160,
                            "height": 160,
                            "formats": {
                                "thumbnail": {
                                    "name": "thumbnail_logo_park_el (1).png",
                                    "hash": "thumbnail_logo_park_el_1_3acd99df65",
                                    "ext": ".png",
                                    "mime": "image/png",
                                    "path": null,
                                    "width": 156,
                                    "height": 156,
                                    "size": 23.26,
                                    "url": "/uploads/thumbnail_logo_park_el_1_3acd99df65.png"
                                }
                            },
                            "hash": "logo_park_el_1_3acd99df65",
                            "ext": ".png",
                            "mime": "image/png",
                            "size": 4.38,
                            "url": "/uploads/logo_park_el_1_3acd99df65.png",
                            "previewUrl": null,
                            "provider": "local",
                            "provider_metadata": null,
                            "createdAt": "2022-05-07T11:26:27.899Z",
                            "updatedAt": "2022-05-07T11:26:27.899Z"
                        }
                    }
                ]
            }
        }
    },
    "meta": {}
}
```


8.15 Get all places

URL	{base URL}/api/places?populate={object} &sort={sortBy} &pagination[withCount]={withCount} &pagination[page]={page}&pagination[pageSize]={pageSize} &pagination[start]={pageStart}&pagination[limit]={pageLimit}
Method	GET
Description	Returns all places
Parameters	object: the name of the nested object(s) to populate in the response. Can use dot notation to further expand nested objects. Can use a wildcard (*) to populate all objects. sortBy: the field to sort the list by page: which page to display (when paginating by page)

	pageSize: the size of the page to display (when paginating by page) pageStart: position of first list entry to return (when paginating by offset) pageLimit: number of list entries to return (when paginating by offset) withCount: boolean to toggle displaying total number of entries (default: true)

Example: /api/places (GET)


```
{
    "data": [
        {
            "id": 22,
            "attributes": {
                "label": "Lime Kiln",
                "description": null,
                "type": "point",
                "createdAt": "2022-05-08T10:06:08.048Z",
                "updatedAt": "2022-05-08T10:06:08.048Z",
                "hex_color": null
            }
        },
        {
            "id": 23,
            "attributes": {
                "label": "Greenhouse",
                "description": null,
                "type": "point",
                "createdAt": "2022-05-08T10:06:32.159Z",
                "updatedAt": "2022-05-08T10:06:32.159Z",
                "hex_color": null
            }
        },
        {
            "id": 24,
            "attributes": {
                "label": "Olive grove",
                "description": null,
                "type": "area",
                "createdAt": "2022-05-10T10:12:07.911Z",
                "updatedAt": "2022-05-11T17:41:06.372Z",
                "hex_color": "#0a0"
            }
        },
        {
            "id": 25,
            "attributes": {
                "label": "Evolutionary garden",
                "description": null,
                "type": "area",
                "createdAt": "2022-05-10T10:17:08.499Z",
                "updatedAt": "2022-05-10T10:33:52.003Z",
                "hex_color": "#a00"
            }
        },
        {
            "id": 26,
            "attributes": {
                "label": "Grassland",
                "description": null,
                "type": "area",
                "createdAt": "2022-05-10T14:51:07.180Z",
```

```
                "updatedAt": "2022-05-23T10:07:13.946Z",
                "hex_color": "#a0a"
            }
        },
        {
            "id": 27,
            "attributes": {
                "label": "Olive grove",
                "description": null,
                "type": "area",
                "createdAt": "2022-05-10T14:56:40.433Z",
                "updatedAt": "2022-05-11T17:41:19.889Z",
                "hex_color": "#0a0"
            }
        },
        {
            "id": 28,
            "attributes": {
                "label": "Grassland",
                "description": null,
                "type": "area",
                "createdAt": "2022-05-10T15:00:30.818Z",
                "updatedAt": "2022-05-23T10:07:21.579Z",
                "hex_color": "#a0a"
            }
        }
    ],
    "meta": {
        "pagination": {
            "page": 1,
            "pageSize": 25,
            "pageCount": 1,
            "total": 7
        }
    }
}
```


8.16 Get a place

URL	{base URL}/api/places/{id}?populate={object}
Method	GET
Description	Returns a place with the specified id
Parameters	id: the unique identifier of the place object: the name of the nested object(s) to populate in the response. Can use dot notation to further expand nested objects. Can use a wildcard (*) to populate all objects.
Example: /api/places/25?&populate=* (GET)	


```
{
    "data": {
        "id": 25,
        "attributes": {
```

```
            "label": "Evolutionary garden",
            "description": null,
            "type": "area",
            "createdAt": "2022-05-10T10:17:08.499Z",
            "updatedAt": "2022-05-10T10:33:52.003Z",
            "hex_color": "#a00",
            "geodata": [
                {
                    "id": 285,
                    "latitude": 35.530824,
                    "longitude": 24.058942,
                    "text": null
                },
                {
                    "id": 284,
                    "latitude": 35.530392,
                    "longitude": 24.059183,
                    "text": null
                },
                {
                    "id": 286,
                    "latitude": 35.530453,
                    "longitude": 24.059387,
                    "text": null
                },
                {
                    "id": 287,
                    "latitude": 35.530907,
                    "longitude": 24.059146,
                    "text": null
                }
            ],
            "images": {
                "data": null
            },
            "comments": {
                "data": []
            },
            "processes": {
                "data": []
            },
            "services": []
        }
    },
    "meta": {}
}
```


8.17 Get all taxons

URL	{base URL}/api/taxons?populate={object} &sort={sortBy} &pagination[withCount]={withCount} &pagination[page]={page}&pagination[pageSize]={pageSize} &pagination[start]={pageStart}&pagination[limit]={pageLimit}
Method	GET

Description	Returns all taxons
Parameters	object: the name of the nested object(s) to populate in the response. Can use dot notation to further expand nested objects. Can use a wildcard (*) to populate all objects. sortBy: the field to sort the list by page: which page to display (when paginating by page) pageSize: the size of the page to display (when paginating by page) pageStart: position of first list entry to return (when paginating by offset) pageLimit: number of list entries to return (when paginating by offset) withCount: boolean to toggle displaying total number of entries (default: true)

Example: /api/taxons (GET)


```
{
    "data": [
        {
            "id": 7,
            "attributes": {
                "label": "fennel",
                "description": null,
                "species": "Foeniculum vulgare Mill.",
                "type": "plants",
                "createdAt": "2022-05-12T06:25:36.914Z",
                "updatedAt": "2022-05-25T18:00:03.327Z"
            }
        },
        {
            "id": 8,
            "attributes": {
                "label": "mastic tree",
                "description": null,
                "species": "Pistacia lentiscus L",
                "type": "plants",
                "createdAt": "2022-05-24T14:44:32.226Z",
                "updatedAt": "2022-05-24T14:44:32.226Z"
            }
        },
        {
            "id": 9,
            "attributes": {
                "label": "weasel",
                "description": "The least weasel varies greatly in size over its range.\nThe body is
slender and elongated, and the legs and tail are relatively short.\nThe color varies geographically, as
does the pelage type and length of tail. The dorsal surface, flanks, limbs and tail of the animal are
usually some shade of brown while the underparts are white. The line delineating the boundary between the
two colors is usually straight. At high altitudes and in the northern part of its range, the coat becomes
pure white in winter.\nEighteen subspecies are recognized.",
                "species": "Mustela nivalis",
                "type": "animals",
                "createdAt": "2022-05-24T21:17:47.057Z",
                "updatedAt": "2022-12-02T00:36:10.044Z"
            }
        },
        {
            "id": 10,
            "attributes": {
                "label": "raven",
                "description": null,
                "species": "Corvus corax",
```

```
                "type": "animals",
                "createdAt": "2022-05-25T17:53:46.692Z",
                "updatedAt": "2022-05-25T18:00:08.070Z"
            }
        }
    ],
    "meta": {
        "pagination": {
            "page": 1,
            "pageSize": 25,
            "pageCount": 1,
            "total": 4
        }
    }
}
```


8.18 Get a taxon

URL	{base URL}/api/taxons/{id}?populate={object}
Method	GET
Description	Returns a taxon with the specified id
Parameters	id: the unique identifier of the taxon object: the name of the nested object(s) to populate in the response. Can use dot notation to further expand nested objects. Can use a wildcard (*) to populate all objects.

Example: /api/taxons/9?&populate=meta_values (GET)


```
{
    "data": {
        "id": 9,
        "attributes": {
            "label": "weasel",
            "description": "The least weasel varies greatly in size over its range.\nThe body is slender and elongated, and the legs and tail are relatively short.\nThe color varies geographically, as does the pelage type and length of tail. The dorsal surface, flanks, limbs and tail of the animal are usually some shade of brown while the underparts are white. The line delineating the boundary between the two colors is usually straight. At high altitudes and in the northern part of its range, the coat becomes pure white in winter.\nEighteen subspecies are recognized.",
            "species": "Mustela nivalis",
            "type": "animals",
            "createdAt": "2022-05-24T21:17:47.057Z",
            "updatedAt": "2022-12-02T00:36:10.044Z",
            "meta_values": [
                {
                    "id": 139,
                    "value": "north america"
                },
                {
                    "id": 138,
                    "value": "europe"
                },
                {
```

```
                    "id": 140,
                    "value": "asia"
                },
                {
                    "id": 141,
                    "value": "20"
                }
            ]
        }
    },
    "meta": {}
}
```


8.19 Get all trips

URL	{base URL}/api/trips?populate={object} &sort={sortBy} &pagination[withCount]={withCount} &pagination[page]={page}&pagination[pageSize]={pageSize} &pagination[start]={pageStart}&pagination[limit]={pageLimit}
Method	GET
Description	Returns the trip list
Parameters	object: the name of the nested object(s) to populate in the response. Can use dot notation to further expand nested objects. Can use a wildcard (*) to populate all objects. sortBy: the field to sort the list by page: which page to display (when paginating by page) pageSize: the size of the page to display (when paginating by page) pageStart: position of first list entry to return (when paginating by offset) pageLimit: number of list entries to return (when paginating by offset) withCount: boolean to toggle displaying total number of entries (default: true)

Example: /api/trips (GET)


```
{
    "data": [
        {
            "id": 5,
            "attributes": {
                "label": "A look at the gorge",
                "description": "this is a beautiful gorge",
                "featured": false,
                "createdAt": "2022-05-24T14:48:55.091Z",
                "updatedAt": "2022-07-22T09:24:31.191Z"
            }
        },
        {
            "id": 6,
            "attributes": {
                "label": "A trip",
                "description": "This is a new trip!~",
                "featured": false,
```

```
                "createdAt": "2022-12-02T00:18:57.827Z",
                "updatedAt": "2022-12-02T00:18:57.827Z"
            }
        }
    ],
    "meta": {
        "pagination": {
            "page": 1,
            "pageSize": 25,
            "pageCount": 1,
            "total": 2
        }
    }
}
```


8.20 Get a trip

URL	{base URL}/api/trips/{id}?populate={object}
Method	GET
Description	Returns the trip with the specified id
Parameters	id: the unique identifier of the trip object: the name of the nested object(s) to populate in the response. Can use dot notation to further expand nested objects. Can use a wildcard (*) to populate all objects.

Example: /api/trips/5?&populate=path (GET)


```
{
    "data": {
        "id": 5,
        "attributes": {
            "label": "A look at the gorge",
            "description": "this is a beautiful gorge",
            "featured": false,
            "createdAt": "2022-05-24T14:48:55.091Z",
            "updatedAt": "2022-07-22T09:24:31.191Z",
            "path": [
                {
                    "id": 488,
                    "latitude": 35.53008710264105,
                    "longitude": 24.05569195747376,
                    "text": null
                },
                {
                    "id": 489,
                    "latitude": 35.53016568375442,
                    "longitude": 24.05605673789978,
                    "text": null
                },
                {
                    "id": 490,
                    "latitude": 35.53030538332148,
                    "longitude": 24.056346416473392,
```

```
                    "text": null
                },
                {
                    "id": 491,
                    "latitude": 35.53037523301381,
                    "longitude": 24.056636095047,
                    "text": null
                },
                {
                    "id": 492,
                    "latitude": 35.53040142663275,
                    "longitude": 24.056990146636966,
                    "text": null
                },
                {
                    "id": 493,
                    "latitude": 35.53041888904063,
                    "longitude": 24.05741930007935,
                    "text": null
                },
                {
                    "id": 494,
                    "latitude": 35.530453813845,
                    "longitude": 24.057977199554443,
                    "text": null
                },
                {
                    "id": 495,
                    "latitude": 35.53075067406825,
                    "longitude": 24.05832052230835,
                    "text": null
                },
                {
                    "id": 496,
                    "latitude": 35.53087291031154,
                    "longitude": 24.058760404586792,
                    "text": null
                },
                {
                    "id": 497,
                    "latitude": 35.530934028363326,
                    "longitude": 24.059264659881595,
                    "text": null
                }
            ]
        }
    },
    "meta": {}
}
```


8.21 Create a trip

URL	{base URL}/api/trips/
Method	POST
Description	Create a new trip and return it

Body: JSON (type shown in typescript interface notation)

{
  data: {
  label: *string*;
  description: *string*;
  path?:{
  latitude?: *number*;
  longitude?: *number*;
  }[];
  observations?: *number[]*;
  commentaries?: *number[]*;
  comments?: *number[]*;
  featured?: *boolean*;
  fav_users?: *number[]*;
  creator?: *number*;
  image?: *number*;
  }
}

8.22 Update a trip

URL	{base URL}/api/trips/{id}
Method	PUT
Description	Update an existing trip and return it
Parameters	id: the unique identifier of the trip
Body: JSON (type shown in typescript interface notation)	

{
  data: {
  label: *string*;
  description: *string*;
  path?:{
  latitude?: *number*;
  longitude?: *number*;
  }[];
  observations?: *number[]*;
  commentaries?: *number[]*;
  comments?: *number[]*;
  featured?: *boolean*;
  fav_users?: *number[]*;
  creator?: *number*;
  image?: *number*;
  }
}	

9. List of figures

10. Glossary of Technical Terms

ACID Compliance: A set of properties (Atomicity, Consistency, Isolation, Durability) that guarantee database transactions are processed reliably.

API (Application Programming Interface): A set of rules and protocols that allows different software applications to communicate with each other.

Augmented Reality (AR): Technology that superimposes computer-generated images on a user's view of the real world, providing a composite view.

Authentication: The process of verifying the identity of a user or system.

Backend: The server-side of an application, responsible for database interactions, authentication, and business logic.

Citizen Science: Scientific research conducted, in whole or in part, by amateur scientists or members of the public.

Client-Side Rendering: Web page rendering performed primarily in the browser using JavaScript.

Cloudflare CDN: A content delivery network service that provides distributed domain name server services and protection from DDoS attacks.

Clustered Architecture: A system architecture where multiple servers work together to provide high availability and load balancing.

CMS (Content Management System): Software application used to create and manage digital content, allowing users without specialized technical knowledge to modify website content.

CORS (Cross-Origin Resource Sharing): A mechanism that allows restricted resources on a web page to be requested from another domain.

CRUD: Create, Read, Update, Delete - the four basic functions of persistent storage.

CSS (Cascading Style Sheets): A style sheet language used for describing the presentation of a document written in HTML.

Data Validation: The process of ensuring data is clean, correct, and useful before it is processed or stored.

DDoS (Distributed Denial of Service): A cyber-attack where multiple systems flood the bandwidth or resources of a targeted system.

Environmental Education: Education about and advocacy for the sustainability of the natural and built environment.

Faceted Search: A technique that allows users to navigate information by applying multiple filters.

Frontend: The client-side of an application, responsible for the user interface and user experience.

Gamification: The application of game-design elements and game principles in non-game contexts.

Geocaching: An outdoor recreational activity where participants use a GPS receiver or mobile device to hide and seek containers, called "geocaches" or "caches."

Geospatial Data: Information that identifies the geographic location and characteristics of natural or constructed features.

GraphQL: A query language for APIs and a runtime for executing those queries with existing data.

Headless CMS: A content management system that manages content creation and storage, but leaves the presentation layer to another technology.

HTML (HyperText Markup Language): The standard markup language for documents designed to be displayed in a web browser.

HTTP (Hypertext Transfer Protocol): An application protocol for distributed, collaborative, hypermedia information systems.

HTTPS (Hypertext Transfer Protocol Secure): An extension of HTTP used for secure communication over a computer network.

IoT (Internet of Things): A system of interrelated computing devices, mechanical and digital machines with the ability to transfer data over a network without human intervention.

JSON (JavaScript Object Notation): A lightweight data interchange format that is easy for humans to read and write and easy for machines to parse and generate.

JWT (JSON Web Token): A compact, URL-safe means of representing claims to be transferred between two parties.

Knex.js: A SQL query builder for JavaScript that makes it easier to build complex database queries.

Leaflet: An open-source JavaScript library for mobile-friendly interactive maps.

Load Balancing: The distribution of workloads across multiple computing resources to optimize resource use and prevent overload.

MariaDB: An open-source relational database management system that is a fork of MySQL.

Metadata: Data that provides information about other data, such as data describing the features of observations in the park.

Microservices: An architectural style that structures an application as a collection of services that are independently deployable and loosely coupled.

Mobile-First Design: A web development approach where the design process starts with designing for mobile devices first.

MVC (Model-View-Controller): A software design pattern commonly used for developing user interfaces that divides the related program logic into three interconnected elements.

Next.js: A React framework that enables server-side rendering and generating static websites.

NGINX: A web server that can also be used as a reverse proxy, load balancer, mail proxy, and HTTP cache.

Node.js: An open-source, cross-platform, back-end JavaScript runtime environment that executes JavaScript code outside a web browser.

OAuth: An open standard for access delegation, commonly used as a way for users to grant websites or applications access to their information on other websites.

ORM (Object-Relational Mapping): A programming technique for converting data between incompatible type systems using object-oriented programming languages.

Paper Prototype: A rough, disposable prototype of a user interface, commonly used for usability testing.

Participatory Design: An approach to design that attempts to actively involve all stakeholders in the design process.

Persona: A fictional character created to represent a user type that might use a site, brand, or product in a similar way.

Personas-based Design: A user-centered design methodology that creates fictitious personas to guide design decisions.

PM2: A production process manager for Node.js applications that helps keep applications alive forever and reload them without downtime.

PPGIS (Public Participation GIS): A subset of participatory GIS that emphasizes involving the public in decision-making processes through GIS technology.

Progressive Web App (PWA): A type of application software delivered through the web, built using common web technologies that is intended to work on any platform with a standards-compliant browser.

Pub/Sub (Publish/Subscribe): A messaging pattern where senders of messages (publishers) do not program the messages to be sent directly to specific receivers (subscribers).

Rate Limiting: A technique used to control the amount of incoming and outgoing traffic to or from a network.

React: A JavaScript library for building user interfaces, particularly single-page applications.

Redis: An in-memory data structure store, used as a database, cache, and message broker.

Responsive Design: An approach to web design that makes web pages render well on a variety of devices and window or screen sizes.

REST (Representational State Transfer): An architectural style for providing standards between computer systems on the web, making it easier for systems to communicate with each other.

Reverse Proxy: A type of proxy server that retrieves resources on behalf of a client from one or more servers.

Scalability: The capability of a system to handle a growing amount of work, or its potential to be enlarged to accommodate that growth.

SEO (Search Engine Optimization): The process of improving the quality and quantity of website traffic to a website from search engines.

Server-Side Rendering (SSR): The process of rendering web pages on the server and sending the fully rendered page to the client.

Socket.IO: A JavaScript library for real-time web applications that enables real-time, bidirectional communication between web clients and servers.

SQL (Structured Query Language): A standard language for storing, manipulating, and retrieving data in relational database management systems.

SSL/TLS (Secure Sockets Layer/Transport Layer Security): Cryptographic protocols designed to provide communications security over a computer network.

Stakeholder: Any person, group or organization that has interest or concern in an organization or project.

Storyboard: A graphic organizer that consists of illustrations or images displayed in sequence for the purpose of pre-visualizing a motion picture, animation, or interactive media sequence.

Strapi: An open-source headless CMS that allows for content type building and API generation.

Taxonomic Data Model: A hierarchical structure for classifying and organizing species and observations within the park application.

Traditional Ecological Knowledge: The body of knowledge, practice, and belief, evolving by adaptive processes and handed down through generations by cultural transmission.

TypeScript: A strict syntactical superset of JavaScript that adds static typing to the language.

UGC (User-Generated Content): Any form of content created by users of a system or service and made available publicly on that system.

UI (User Interface): The space where interactions between humans and machines occur.

Use Case: A list of actions or event steps typically defining the interactions between a role and a system to achieve a goal.

User Experience (UX): A person's emotions and attitudes about using a particular product, system, or service.

User-Centered Design: A design process in which the needs, wants, and limitations of end users of a product are given extensive attention at each stage of the design process.

VGI (Volunteered Geographic Information): The harnessing of tools to create, assemble, and disseminate geographic data provided voluntarily by individuals.

VPS (Virtual Private Server): A virtual machine sold as a service by an Internet hosting service.

Web Mapping: The process of using maps delivered by geographical information systems on the internet.

WebSocket: A computer communications protocol, providing full-duplex communication channels over a single TCP connection.

Wireframe: A visual guide that represents the skeletal framework of a website or app.