Campus Cats Mobile Application

JIC 3341

Matthew Pendarvis, Robert Zhu, Dragos Lup, Amulya Panakam, William Akins

Client: Campus Cats

Repository: https://github.com/willakins/JIC-4331-ScrapCats.git

Table of Contents

Table of Figures

Terminology

API (Application Programming Interface): A set of rules and protocols that defines how two applications communicate and interact with each other.

Back-end: The portion of an application that operates behind the scenes and is not directly accessible to users.

CSS (Cascading Style Sheets): A stylesheet language used to define the visual presentation and layout of webpages.

Cat-alog: A wiki containing detailed information about the cats residing on campus.

Front-end: The user-facing part of an application that allows interaction through a graphical interface.

Geolocation: The method of identifying the physical location of a device using various technologies.

Geo-tag: A label added to a digital file that specified the geographic location where it was created.

JSON (JavaScript Object Notation): A format for exchanging data, using human-readable text to represent objects and arrays.

Native Application: An application installed directly onto a device, as opposed to being accessed via web browser.

React Native: A JavaScript framework used to build mobile user infaces across platforms.

TypeScript: A programming language that extends JavaScript by adding strong typing and object-oriented features.

SQL: A programming language used to manage, manipulate, and query data in relational databases.

Node.js: A JavaScript runtime environment that allows developers to tun JavaScript code server-side, enabling the development of scalable and fast network applications.

Introduction

Background

The Georgia Tech Campus Cats club is dedicated to managing and supporting the welfare of the stray and feral cat population on campus. The club's goals include promoting awareness, fostering community engagement, and ensuring the health and safety of the cats. However, the lack of a centralized system for logging sightings, tracking individual cats, and coordinating communication has resulted in fragmented caregiving efforts and inefficiencies. This gap affects both the cats, which may receive inconsistent care, and the volunteers, who often duplicate work or face challenges in maintaining continuity.

To address these challenges, the proposed mobile communication app offers a comprehensive solution tailored to the needs of the Georgia Tech community. The app combines essential features such as geo-tagging, a centralized database, messaging, and announcements to improve coordination and streamline operations. By fostering a collaborative environment, the app ensures that all campus cats receive timely and organized care, while volunteers can efficiently manage and share their efforts.

Key Features

The **Cat-alog** serves as the heart of the application, offering a centralized repository of campus cat profiles. Users can browse through a list of cats, each accompanied by a detailed profile containing information about their physical characteristics, health history, and recent sightings. To enhance community engagement, the app allows users to upload photos and contribute updates to each profile, creating a dynamic and ever-evolving knowledge base. This feature ensures that every cat on campus is accounted for and fosters a sense of connection and responsibility among club members.

The **geo-tagging and mapping** feature empowers users to report cat sightings in real-time by uploading geo-tagged photos and attaching notes about the cat's condition or location. A living map displays these sightings, offering a comprehensive view of the campus cat population at any given time. This functionality not only enhances the monitoring efforts of volunteers but also aids in the coordination of feeding schedules, medical care, and other interventions, making it a critical tool for effective population management.

An **in-app messaging system** provides a seamless platform for communication between members, administrators, and volunteers. Users can engage in direct messaging or group chats to coordinate feeding schedules, arrange medical appointments, or discuss upcoming events such as adoption drives. This feature ensures real-time collaboration and allows members to stay connected, fostering a stronger and more coordinated community dedicated to the welfare of campus cats.

The **announcements feature** allows administrators to share important updates with all users, such as event notifications, emergency alerts, or changes to care plans. Members receive

these updates through push notifications, ensuring they stay informed about ongoing initiatives or urgent matters. By centralizing announcements within the app, this feature keeps users aligned with the club's activities and enhances overall communication efficiency.

To encourage broader participation, the app includes **donation and volunteer management tools**. Donors can contribute funds or supplies while tracking the impact of their contributions through detailed updates. Additionally, volunteers can manage their roles by signing up for shifts, feeding schedules, or other tasks directly within the app. This feature not only ensures transparency in resource allocation but also simplifies the process of engaging new members and sustaining volunteer efforts.

Finally, the app offers **integration with existing systems**, specifically the Campus Cats website, to synchronize data and maintain consistency across platforms. By leveraging the club's existing resources, the app minimizes redundancy while significantly expanding functionality. This integration ensures that users can seamlessly transition between the website and the app while benefiting from the enhanced features provided by the mobile platform.

These features work in tandem to create a robust and user-friendly platform, empowering members to effectively manage the welfare of the campus cat population. The app's thoughtful design and functionality ensure that all stakeholders—students, faculty, and volunteers—can contribute meaningfully to the mission of the Georgia Tech Campus Cats club.

Document Summary

System Architecture

Our system is a mobile application that will run on both IOS and Android devices. As a general-purpose application for the Georgia Tech club Campus Cats, our system requires a robust architecture that can handle many types of requests (uploading images, sending messages, viewing maps) while ensuring each component is self-composed. Our system can be broken into three layers: the presentation layer which the user sees and interacts with; the business layer which controls logic and control flow of the user's inputs; and the storage layer which holds persistent data like user profiles. As this mobile application is meant to be used by Georgia Tech students and faculty, we developed a platform agnostic system in case not everyone had the same mobile software. A layered architecture has the benefit of low coupling among components, allowing for parallel development of both front-end and back-end components during sprints.

Component Design

The component design section details the modular elements of the system and their interactions. It defines core components like the Cat-alog module, Messaging system, Announcements manager, and Sighting map viewer, specifying their roles and interactions. For example, the Cat-alog module interacts with the storage layer to fetch cat data and user-uploaded images while displaying them via the presentation layer. The design focuses on

ensuring low coupling and high cohesion, enabling seamless integration and ease of maintenance.

Data Storage Design

This section describes how data is structured and stored to meet the application's requirements. It outlines the database schema for key entities such as user profiles, cat information, images, chat messages, and announcements. For instance, the cat-alog will have a database table with fields for cat ID, name, description, image URLs, and last sighting location. The section highlights the use of a cloud-based relational database for structured data and a separate storage solution for media files (e.g., Firebase Storage or AWS S3). Emphasis is placed on data integrity, security, and efficient querying to ensure responsive app performance.

UI Design

The UI design section focuses on the application's user interface, detailing wireframes and user interaction flows for key features. It describes the layout of the **Cat-alog** screen, including a scrolling pane with cat icons that link to individual cat profiles, which contain details and user-uploaded images. The design prioritizes usability, with a clean and intuitive interface that aligns with Georgia Tech branding. Key design principles such as responsive layouts, clear navigation, and accessibility considerations (e.g., contrast, font size) are emphasized to enhance the user experience across different devices.

System Architecture

Introduction

Our system is a mobile application that will run on both IOS and Android devices. As a general-purpose application for the Georgia Tech club Campus Cats, our system requires a robust architecture that can handle many types of requests (uploading images, sending messages, viewing maps) while ensuring each component is self-composed. Our system can be broken into three layers: the presentation layer which the user sees and interacts with; the business layer which controls logic and control flow of the user's inputs; and the storage layer which holds persistent data like user profiles.

Rationale

As this mobile application is meant to be used by Georgia Tech students and faculty, we developed a platform agnostic system in case not everyone had the same mobile software. A layered architecture has the benefit of low coupling among components, allowing for parallel development of both front-end and back-end components during sprints. To enforce the security of Campus Cats' data, a permission system will be set in place based on the authentication service; this will allow only admin users to have full access to all information.

Static System Architecture

This diagram presents the high-level interactions between major components in our system.

Diagram

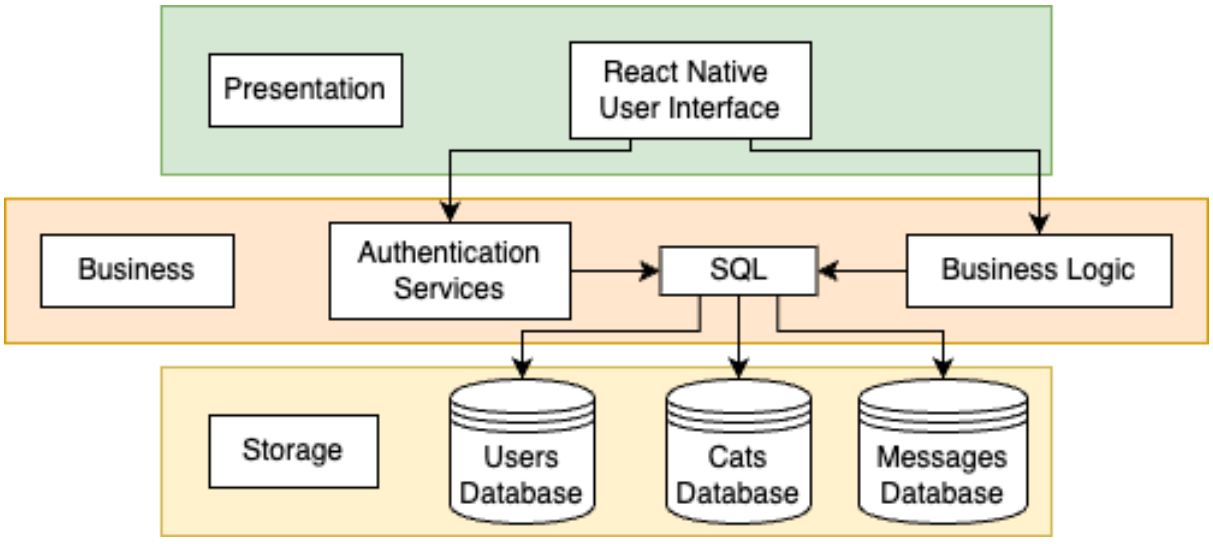

Figure 1: Static System Diagram

Description

Our application has 3 separate layers, the first being the presentation layer. The presentation layer is entirely made in React Native and is the primary way for the user to interact with the system through the business layer. This layer is connected to and will be sent information

from two components from the business layer. The information is then shown to the user who can then interact with the app as normal. The business layer has two main components, the authentication services and business logic, which will communicate using SQL to get access to the storage layer. The authentication services will handle security processes and connect with the user's database. The business logic will perform all other functions of the app, such as managing the cats, users, messages, and images. Both will be using SQL to communicate with the firebase storage. It is vital that these two layers remain separate and easy to communicate between because after we leave this project to campus cats, it is likely they might have to find a different data solution, and we would like to make the transfer as easy as possible. The storage layer contains 3 databases. The user's database will store user information, like passwords and security level. The Cats database manages cats, their images, and information pertaining to the cats. The messages database will store messages that users send to each other.

This structure is very modular overall, and it will be easy to change functions of one part of the app while keeping the rest intact, for example if the clients would like a new front end later on, or if they'd like a website along with an app, it would be easy to grab information from the storage layer using a separate business layer.

Dynamic System Architecture

The system sequence diagram (SSD) in Figure 2 demonstrates the interactions between the components of our application when a user makes a report of a new cat sighting.

Diagram

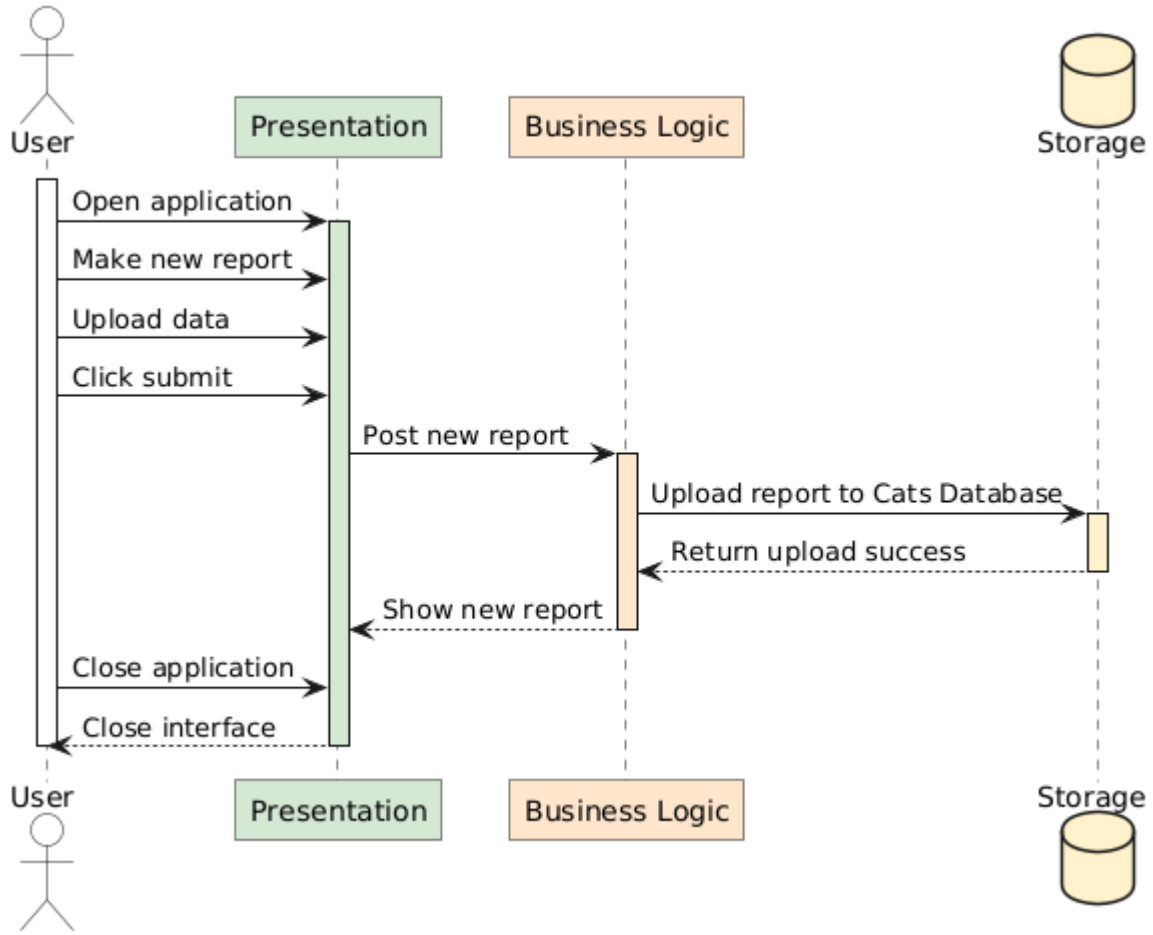

Figure 2: Dynamic System Diagram

Description

Our application's most important feature is allowing users to report new sightings of cats across campus. The user creates a new report by uploading a picture and any relevant information, and the application then uploads the report to the database. Once the new report has been uploaded to the database, the application presents the new report to the user via the interface so they can confirm that the creation of the new report was successful.

We chose a system sequence diagram to depict the system, as it focuses only on the interaction of the different systems and not the components that make up each system.

Component Detailed Design

Introduction

Our application used a three-layered architecture system, consisting of the User Interface (UI), Business Logic, and Database. This modular design ensures scalability, security, and efficient communication between components. The UI, developed in React Native, provides an interactive user experience, while the Business Logic, implemented in Node.js, serves as the intermediary between the UI and the Database, which is structured in SQL to manage and store data securely.

To illustrate how these layers interact, we present two key diagrams:

- The Static Component Diagram: This diagram provides an overview of how different components within our system communicate. It depicts how the frontend (UI) interacts with backend (Business Logic and how the backend, in turn, connects with the database to retrieve, store, and update data. Key functionalities such as user authentication, reporting cat sightings, messaging, and announcements are mapped within this architecture.
- The Dynamic Component Diagram: This diagram captures the real-time interactions between components as users engage with the application. It details how user actions, such as reporting a cat sighting or retrieving existing reports, trigger backend processes and database queries. Specifically, it highlights the flow of data when users navigate the campus map, create reports, or retrieve stored information from the database.

By maintaining a clear separation between these layers, our architecture ensures flexibility, security, and efficiency. Future modifications, such as switching databases or hosting solutions, can be seamlessly implemented without disrupting the core functionality of the system. Further below, this document explores how static and dynamic components interact to create a robust, user-friendly experience.

Diagram

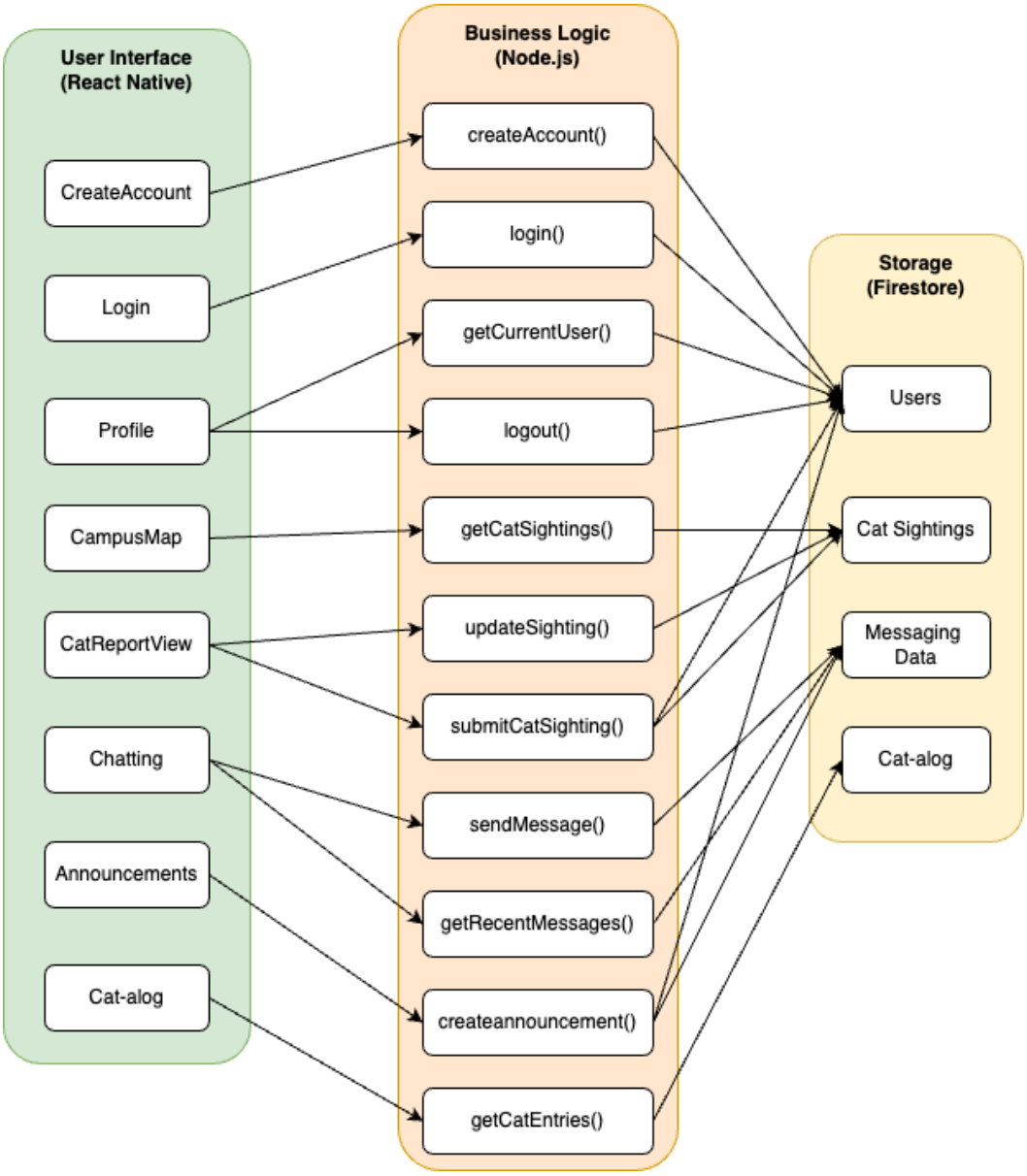

Figure 3: Static Component Diagram

Description

Our application has three layers of architecture: the User Interface, the Business Logic, and the Databases. As explained previously, the UI will be in React Native, and will communicate with the Business Logic, which will be coded in Node.js. These will be referred to as the frontend and backend, respectively. The backend will then communicate with the databases, which is implemented in SQL. The Static Component Diagram specifies which functions will communicate between which of the layers of architecture. The backend will be running on a render server; however, this can be changed later to the server of the client's choice.

When a user decides to create an account, the frontend will send a request to the backend to run createAccount(), where the backend will check for username availability and password security, and if everything works out, then the username will be added to the

database. It is important that all database functionality happens on the backend for security reasons. Login is a similar process but check for availability we will be checking if the username exists with the password specified. Then the user information from the database will be sent to the front end. The security functionality will be created through firebase, for its ease of use and functionality with google accounts.

The Campus Map will call on the default screen and will make a call to the backend to ask for recent posts. getCatSightings() will access the cat database and return a selected recent number of cat sightings, depending on an algorithm. To report a cat, similar calls will be made to the backend through updateSighting() and submitCatSighting(). The backend will then access the database and store new data given to it by the frontend. The catalog will match this functionality but rather than access the cat's database, will access the catalog database.

The messaging feature will work similarly to the previous two, using the backend to access database information at the front end. However, the backend will be managing the messages slightly differently. When the user submits a message, the backend will send the message to its database and the frontend users will have a listening signal active, which will be notified of the change in the message database, of course only if the chat includes them. Each message group set will have its own dataset in the messaging data, and each user in user data will have a list of the chats they have, for easy access, rather than sorting through all the groups. The messaging functionality will also use firebase. When creating an announcement, the backend will ensure the user is an admin first, and then update that user’s announcements made, and create the new announcement in the messaging database.

It is important to note that the 3 layers are entirely separate, and modular if the client wishes to change functionality. For example, the databases can be changed from firebase to another service, or rather than run the backend on Render, a custom solution can be implemented.

Dynamic

Diagram

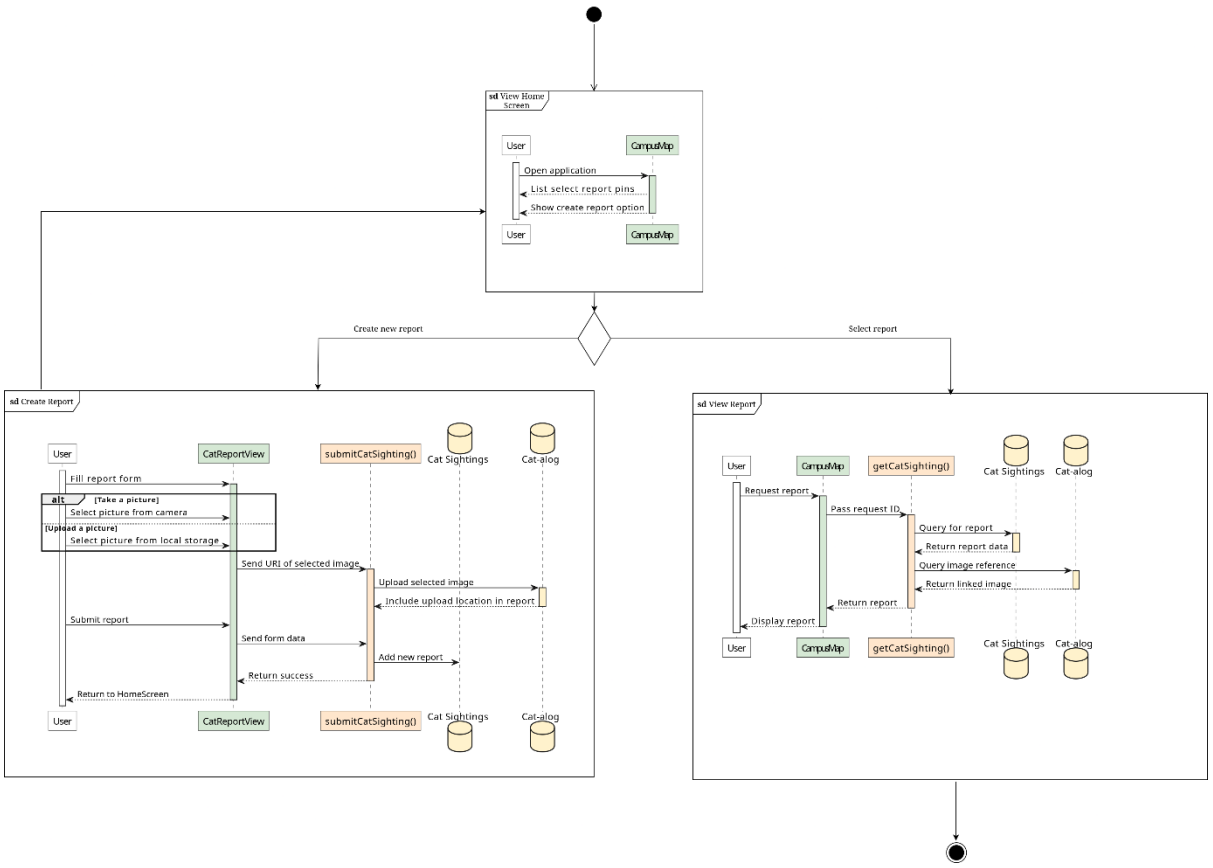

Figure 4: Dynamic Component Diagram

Description

Figure 4 is an Interaction Overview Diagram depicting the process of making or viewing reports of cat sightings. The diagram was chosen as it succinctly shows how several different components of the system collaborate during the execution of a common workflow. From the campus map, the user can either make a new report by pressing the report button or view an existing report by selecting one of the pins on the map. If the user chooses to create a report, they are taken to a report screen, where they can fill out any information and take or submit a picture to complete the report. This information is then stored in the database with the rest of the reports, while the image is submitted to the Cat-alog storage with the rest of the images. Afterwards, the user is guided back to the campus map. If the user is on the campus map and chooses to view an existing report, they are taken to the cat sighting screen. On this page, a call is made to the backend to get the report data which is then displayed to the user.

Data Design

Introduction

The Campus Cats Database is an essential component of the Campus Cats Mobile Application, designed to help Georgia Tech students and faculty manage and support the welfare of stray and feral cats on campus. The database facilitates seamless data storage,

retrieval, and exchange, ensuring volunteers and administrators can effectively track, manage, and care for the campus cats.

This section provides a comprehensive overview of how the database is structured, used, and secured within the application. It includes details about database design, file usage, data exchange protocols, and security considerations to ensure reliability and efficiency.

Database Design

The Campus Cats application uses Firebase's NoSQL document-based database Firestore, designed to efficiently store and manage unstructured data such as cat profiles, user reports, and announcements. The application is built to handle multiple data types dynamically, ensuring easy scalability and flexibility. Each document corresponds to an entity such as a cat profile, user, report, or message.

- Cat Collection:
  Each cat profile is stored as a JSON document containing details such as:


```
{
  "id": "cat456",
  "name": "Whiskers",
  "firstSeen": "2023-07-01",
  "healthStatus": "Healthy",
  "feedingSchedule": "Twice Daily",
  "images": ["img1.jpg", "img2.jpg"],
  "volunteers": ["user123", "user456"],
  "description": "Orange tabby"
}
```


- User Collection:
  Stores user profiles and their roles in the application. Authentication is handled through Firebase Authentication and Georgia Tech's Single Sign-On (SSO), eliminating the need to store passwords.
- Sighting Reports Collection:
  Contains records of cat sightings submitted by users, with locations and images.
- Announcements Collection:
  Stores administrative announcements, ensuring efficient communication among volunteers.

Data Organization and Relationships

The relationships between data entities are maintained using references instead of foreign keys (typical in SQL databases). For instance:

- Each cat document stores an array of volunteers who have interacted with it.
- Each sighting report links back to the catID it belongs to.

The following diagram illustrates the logical structure of the database:

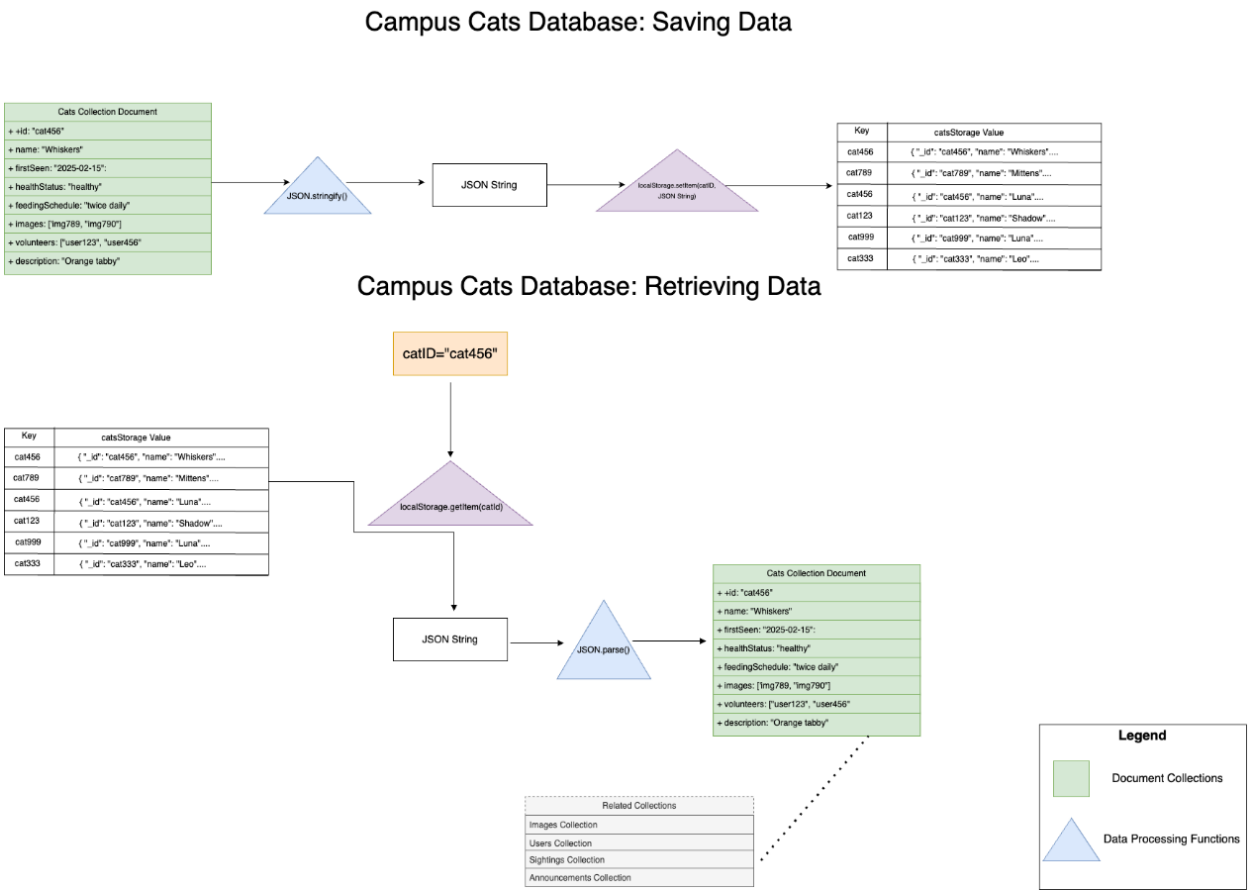

Figure 5: Data Design Diagram

File Use

File Documentation

The Campus Cats application utilizes Firebase storage to handle large media files. The following files are stored permanently:

- User-uploaded cat sightings (e.g., JPEG/HEIC images)
- Profile pictures
- Cat-alog images

- Images for announcements

All images and documents are stored in Firebase Storage, allowing efficient retrieval when users browse through the catalog or reports. The application pulls images dynamically from the Firebase storage bucket, ensuring optimal performance without bloating the database.

Data Exchange

Data is sent to Firestore as JSON data and stored as documents. The mobile application interacts with Firestore using Firebase's SDK, ensuring secure and efficient communication.

- Cat sighting reports, user profiles, and messages are retrieved via Firestore queries.
- New sightings, messages, and reports are added using Firestore document writes.
- Updates to existing records (e.g., feeding schedule changes) are performed through Firestore document updates.

All communications occur over HTTPS to prevent unauthorized access or data interception.

Security Considerations

The Campus Cats application follows strict security measures to protect user data and cat records:

User Authentication

- Georgia Tech Single Sign-On (SSO) Integration:
  The application integrates with Georgia Tech's SSO system, eliminating the need for storing user passwords. Only verified users with valid university credentials can access the system.
- Secure Credential Storage:
  Although the application does not store passwords (due to SSO), any user-related authentication tokens are stored securely using industry-standard hashing and salting techniques through Firebase Authentication.

Data Protection

- End-to-End Encryption:
  - All sensitive user information is encrypted before being stored in the database.
  - HTTPS encryption is enforced for all data exchanges.
- Access Control:
  - Only administrators can post announcements or modify system-wide settings.
  - Volunteers have restricted access to updating cat profiles and submitting reports.

Privacy Considerations

- No personally identifiable information (PII) is stored beyond usernames and roles.
- No payment processing is involved, eliminating risks associated with financial transactions.

By incorporating these security measures, the Campus Cats application ensures user safety, data integrity, and compliance with best security practices.

UI Walkthrough

Introduction

The Campus Cats Mobile Application is designed to provide volunteers and administrators at Georgia Tech with a user-friendly interface for tracking, managing, and supporting the welfare of stray and feral cats on campus. The following walkthrough demonstrates the key screens users interact with, showcasing intuitive navigation, clear information hierarchy, and adherence to design heuristics for usability.

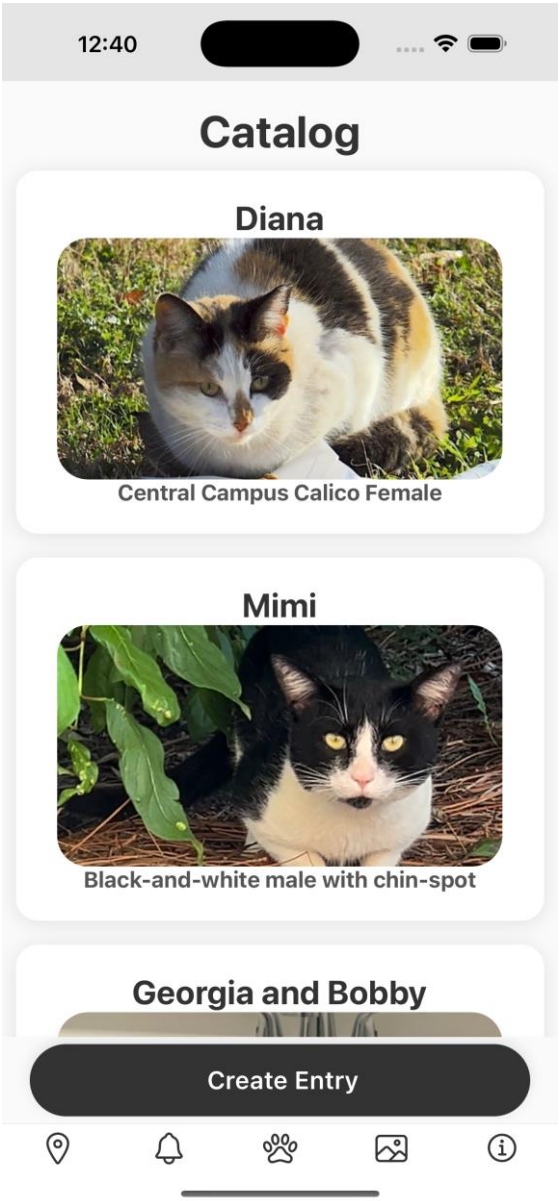

Figure 6: Cat-alog screen

Description: This screen presents a scrollable list of cats registered in the app's Cat-alog. Each

entry includes an image, name, and a brief description. Users can tap on any cat to view its detailed profile.

Heuristic Analysis:

- **Visibility of System Status:** Users can easily see which cats are listed and their status or notes (e.g., "happy… for now").
- **Recognition Rather Than Recall:** Photos and names make it easy to recognize cats instead of remembering their IDs or characteristics.
- **Aesthetic and Minimalist Design:** The layout is clean, using whitespace and visual hierarchy to avoid clutter.

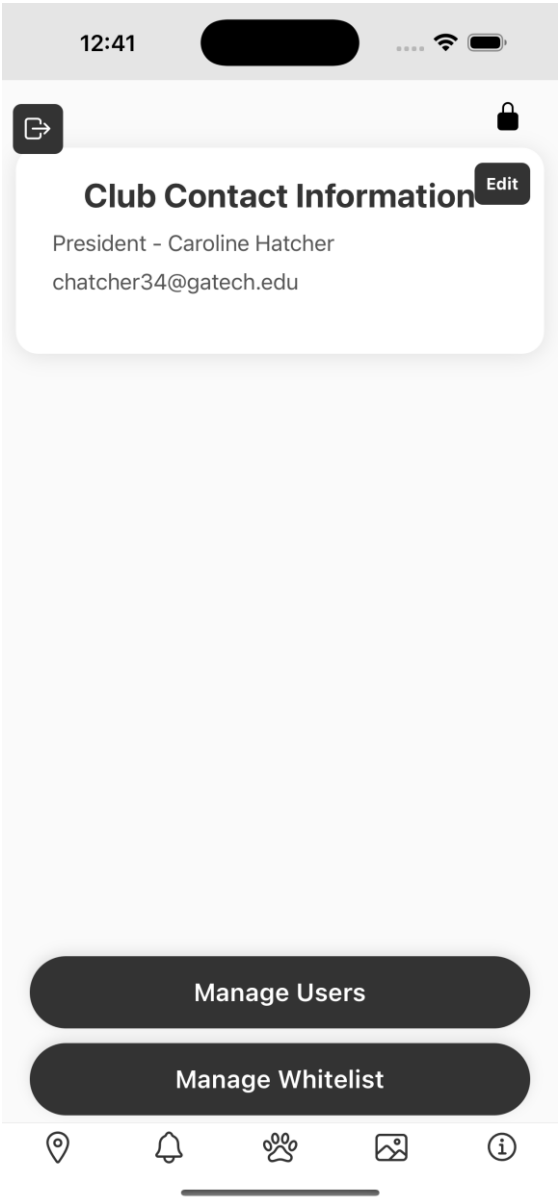

Figure 7: Club Contact Information Screen

Description: Accessible only to administrators, this screen allows for the editing of club contact information, managing users, and maintaining a list of authorized contributors.

Heuristic Analysis:

- **User Control and Freedom:** Admins can make and undo changes to settings like user permissions and contact details.
- **Match Between System and Real World:** Labels such as “Manage Users” and “Edit” use familiar language that reflects real-world tasks.
- **Error Prevention:** Restricted access ensures only admins can perform critical changes, reducing risk of misuse.

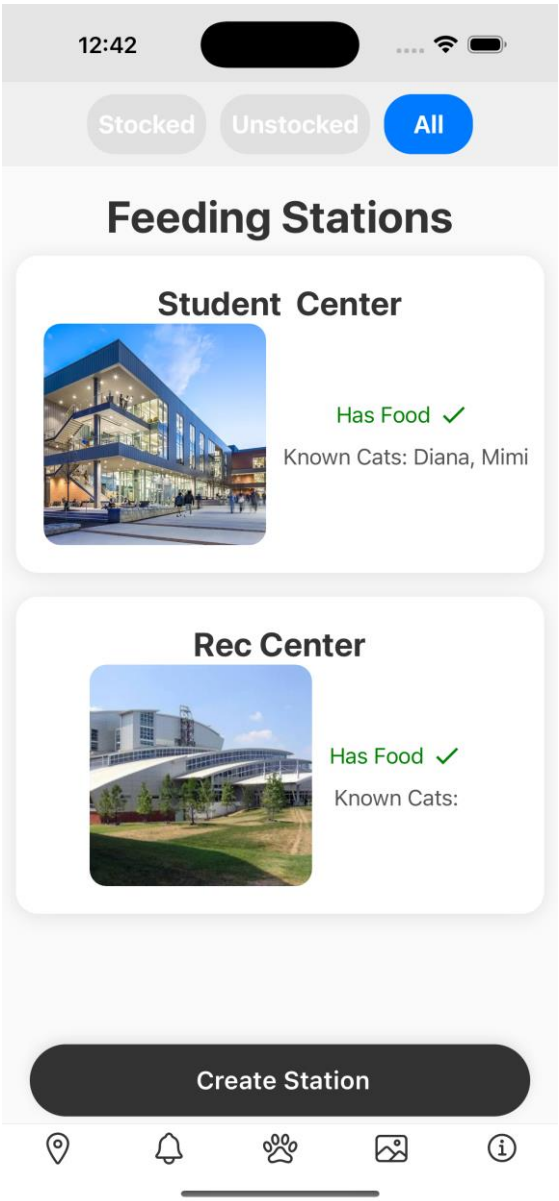

Figure 8: Feeding Station Screen

Description: This screen lists all known feeding stations, each with a photo, name, food status indicator, and list of known cats that use the station.

Heuristic Analysis:

- **Feedback:** The green "Has Food" checkmark clearly communicates the station's current status.
- **Consistency and Standards:** Uses common UI patterns like image cards and labels to maintain uniformity.
- **Flexibility and Efficiency of Use:** Volunteers can quickly confirm food status or update it with one tap.

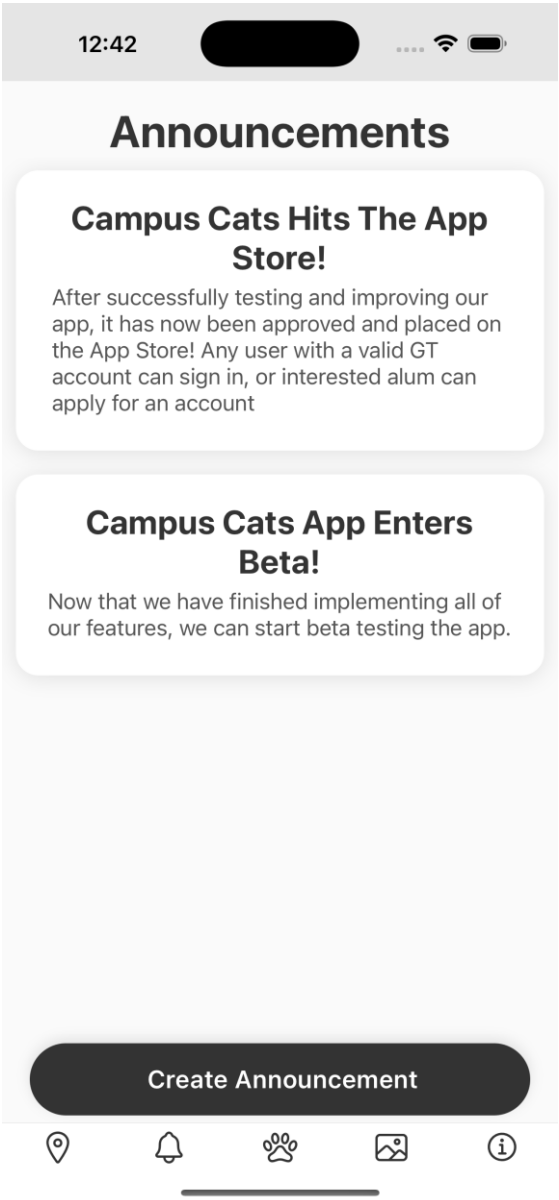

Figure 9: Announcements Screen

Description: This screen allows users to view announcements made by administrators, such as new cat sightings or care updates.

Heuristic Analysis:

- **Consistency and Standards:** This screen follows the same card-style UI as the catalog and feeding stations.
- **Help Users Recognize, Diagnose, and Recover from Errors:** Clear labels and simple structures prevent confusion.
- **Minimalist Design:** Focus is on important announcements only, keeping distraction to a minimum.

Appendix

Name:	Role/Contributions	Contact
Matthew Pendarvis	UI development, React Native implementation, cat catalog design	mpendarvis3@gatech.edu
Robert Zhu	Database schema design, Firebase integration, system testing	arcwand@gatech.edu
Dragos Lup	Backend architecture, Node.js services, API development	dlup3@gatech.edu
William Akins	Project coordination, system architecture, integration between layers	wakins6@gatech.edu
Amulya Panakam	UX research, UI heuristic evaluation, design documentation	apanakam7@gatech.edu