NUS Cats🐱

Find your favourite cats!

Group 11
Tan Kang Liang
Huang Weijie
Lim Jin Hao
Let Yan Wen Dominic

https://nuscats.herokuapp.com

Table of Contents

Milestone 0: Problem Description

Describe the problem that your application solves

Our application aids NUS Cat Cafe in caring for community cats on the NUS Campus.

NUS Cat Cafe is a group in NUS which takes care of the well-being of cats within campus. Their work includes feeding cats on a daily basis, ensuring cats that live on campus are neutered, and sending cats to the vet if needed.

From our interview with the NUS Cat Cafe members, we have identified the following pain points:

1. During the feeding process, feeders would roam around the general vicinity where a particular cat would usually be found. Sometimes, finding a cat can be difficult, and members would sometimes skip feeding for the day if the cat cannot be found. Thus, feeding is not only time consuming, it is also not guaranteed.
2. Reporting of new cats and emergencies (incident reporting) are split across different channels. NUS Cat Cafe receives messages on Facebook, Instagram, Email as well as through a personal contact number. This reduces the visibility of such alerts since only a few members would have access to all of these channels.

NUSCats aims to expedite the feeding process as well as streamline incident reporting through a common channel.

Milestone 1: Application Description & Mobile Cloud Computing

Describe your application and explain how you intend to exploit the characteristics of mobile cloud computing to achieve your application's objectives, i.e. why does it make the most sense to implement your application as a mobile cloud application?

Our application aims to make feeding more efficient, by providing the last-seen location of cats. Specifically, it crowd-sources the locations of the cats in NUS. This is done by allowing users to take a photo of the cat and upload the sighting on the app. The app detects the current location of the user and updates the location of the sighted cat on a map. From such sightings, NUS Cat Cafe members would be able to narrow down the current location of the cats, which would make locating the cats for feeding a lot easier.

For our solution to work, it is a precondition that it should be convenient and portable. Since our application is centered around location service and uploading of cat sighting pictures, it must be convenient for users to take photos of cats and also make the cat's location known. In that sense, it would make the most sense to implement the solution as a mobile application, since almost all smartphones come with a camera and would also have built-in GPS for location tracking.

To streamline incident reporting, the application has to be easily accessible. A website that does not require users to download would be the best choice for our application. Furthermore, since incidents can be reported at any time of the day, it would be beneficial for NUS Cat Cafe members if they were able to obtain a native feel even on their mobile devices. Hence, we are inclined to develop NUSCats as a progressive web application that fits naturally on a mobile device.

Milestone 2: Target Audience & Marketing

Describe your target users. Explain how you plan to promote your application to attract your target users.

Our application is targeted at both NUS Cat Cafe members and the cat lovers within the general NUS student population.

We have approached NUS Cat Cafe members to understand the problems they face and the key features that they would like to see in the application so that we may develop an application targeted at their needs. Thus, since we have a rather clear picture of the needs of the members, when the application is ready, we will convince them to use the app by packaging the benefits of the application as such:

1. Makes feeding more efficient (by cutting down on the time spent locating the cats)
2. Provides an interactive and fun way of reaching out to the general NUS student population (possibly for recruitment purposes)
3. Provides a platform for managing incident reporting
4. Provides a channel for communicating with cat lovers in the NUS community

For the other users, we are specifically targeting cat lovers in the NUS student population. In order to reach such groups of students, we will make use of social media, such as Facebook and Instagram, or chat applications, such as Telegram, to promote our application. These social media platforms are most popular among University students.

Our marketing efforts will focus on the “social” aspect of the application, which we believe would be the main motivation for such users to utilize the app. Our application will have a “feed” that displays all sightings, along with their description, which mimics the idea of social media platforms. In our case, the application can be marketed as an application to foster a community of cat lovers within NUS. The idea of being part of a tightly-knit community would motivate our target users to utilize the application.

Bonus Milestone: Primary key of *home_faculties*

What is the primary key of the home_faculties table? (Not graded)

- Primary key: (matric_no, faculty)

Milestone 3: Entity-Relationship Diagram

Draw an Entity-Relationship diagram for your database schema.

The Entity-Relationship diagram above captures the information stored in our Postgres database. Primary keys are shown in bold while relationships are specified by the connections from one entity to another.

We have also used some custom types such as the following:

1. Point
    a. Latitude and Longitude coordinates for a location from PostGIS plugin
2. UniversityZone
    a. An enum that indicates a particular region within NUS.
    b. Some examples are “Computing”, “Arts”, “UTown” and many others as denoted by NUS Cat Cafe
3. SightingType
    a. An enum to indicate if this is an emergency report or an everyday cat sighting
4. RoleType
    a. An enum indicating if the user is an administrator (from NUS Cat Cafe) or a normal user

Our Users table contains all necessary information needed to authenticate and authorize application users. We have decided to keep this separate from the Profiles table which stores additional information regarding user personalization due to a possible increase in data that needs to be stored in the future. This way, our authentication and authorization logic is unaffected when we need to update users' profiles to include their course of study for example. There is a one to one relationship between Users and Profiles.

The Cats table contains some of the information publicised by NUS Cat Cafe on their original website. We have ported most of these details as well as useful information that might be needed to be kept by members such as whether or not a cat is currently neutered.

Finally, our CatSightings table which contains records of cats spotted by people. Users can choose to create the sightings as an emergency report or just as a normal cat sighting. CatSightings have a many to one relationship with Users as well as a many to one relationship with Cats. When a sighting is reported, we also conduct a reverse geocoding search to obtain a human readable location that is stored as the location’s name.

Milestone 4: REST API Alternatives & Choice

Explore one alternative to REST API (may or may not be from the list above). Give a comparison of the chosen alternative against REST (pros and cons, the context of use, etc.).

	REST	gRPC
Speed of Development	Fast prototyping with JSON support available in virtually all languages	Depends on external code generation tools that need to be run that complicates local development setups across team members.
Transfer Protocol	HTTP 1.1 Only support for typical client-response scenarios where the client makes a request and server responds with a single response	HTTP 2.0 In addition to unary requests, there is support for streaming communication. Also supports “server-push” where information not requested by the client can still be sent by the server.
Serialization	JSON Human readable strings that are easily understood. However, there is an inability to share entities across different languages. Changes made to one will not be propagated to all clients who use them.	Protocol Buffers Binary format that requires the protobuf definition for deserialization. Protobuf definition allows for code generation for a wide range of programming languages.
Error Handling	Use of HTTP status codes but without a standardised format for custom messages	gRPC has their own status code mappings for scenarios similar to HTTP, also includes an optional string message for more details
Browser Support	Supported by most modern-day browsers such as Chrome and Safari which are used in mobile devices	No native support for HTTP2 on browsers, requires the protocol to be built over HTTP 1.1 or through proxy servers

Between REST and your chosen alternative, identify which might be more appropriate for the application you are building for this project. Explain your choice.

From the comparison, it can be seen that gRPC would allow for a more flexible and less error-prone development as protobuf schema files allow for portability to multiple languages and the dependency on such a schema allows schema changes to be propagated to code that uses it. Ultimately, our team decided on REST due to the ease of setting up and the gentle learning curve since we were only given about three weeks for the assignment. The choice of a singular language (Typescript) also meant that some of the benefits of gRPC are lost for our use case. Browser support is also of utmost concern to us and REST is the clear winner in that case.

Milestone 5: API Documentation

Design and document all your REST API.

Link to Swagger: https://nuscats.herokuapp.com/api/

Organisation

Each REST API documentation is titled with their respective request method and relative url (relative to https://nuscats.herokuapp.com/). Within each API documentation, the schema of the request parameters, query and body are shown. The response success code, data and (relevant) error codes for each REST API are also documented and shown,

Sections

The APIs are logically organised into sections (or modules) to show the distinction between each group of API calls. Each API calls have a short description of the expected

A summary of each section is shown here:

Section	Summary
Server	Endpoint used to check the status of the server
Authentication	Endpoints that handle all authentication related calls, from logging in and out to changing password and username
Profiles	CRUD functionalities for user profile Features authenticated endpoints that handles request for getting user profile information, updating or deleting user profile
Cats	CRUD functionalities for cats Features public endpoints for querying cat information Provide admin-authenticated endpoints to create, update or delete cats
Sightings	CRUD functionalities for cat sightings Features public endpoints for querying cat sightings based on query params and filters or creating new

	sightings. (The GET / endpoint returns a paginated response.) Provide authenticated endpoints for admin or the respective sighting creator to update or delete sightings
Uploads	GET requests to obtain signed url to upload images to S3 Rationale: To eliminate the need to send possibly large image data from client to server
Notifications	POST endpoints to subscribe or unsubscribe users to a specified notification topic Authenticated endpoint for admin to push a notification to all users subscribed to the specified notification topic

Also, explain how your API conforms to the REST principles and why you have chosen to ignore certain practices (if any). You will be penalised if your design violates principles for no good reason.

REST principles has 6 architectural constraints:

REST Principles	**Conformations**
Uniform interface	Resources are made available using intuitive URI standards. For example, the “/profile/:id” URI identifies with the profile of the indicated user. The URIs are also logically group such that “/cats” fetches cats information and “/sightings” fetches sightings information JSON is used as the common resource representation for the API responses. For paginated requests (eg. GET /sightings), links are provided which points to the first, previous, next and last page.
Client-server	The REST API conforms to a client-server architecture. Client requests information from the server and displays it to the user. The server manipulates information and stores data. Our client frontend only makes relevant API calls to the backend server. As long as the resource URI remains the same, the client can make API calls to the server. This way, the client and server can evolve separately without any dependency on one another.
Stateless	Our server does not store any user session information or user HTTP request history. When a user logs in, a JWT access token is provided to the user and stored in cookies. When a user subsequently makes an authenticated or authorized operation, the JWT access token provides the necessary authentication and authorization details for the server to verify the user. The server does not track the user’s session information. Our use of pagination also demonstrates the stateless nature of our server. Client’s request contains all relevant pagination information in the query, such as page number and page size. This provides enough information for the server to provide the data for that current page, without

	“remembering” the client’s browsing status.
Cacheable	For NUSCats, caching is done on the client-side to improve performance and provide some degree of offline functionality, which is important for a PWA. Our server also does not handle any resource heavy computations and hence implementing an external cache does not provide much benefit. As we use Heroku Postgres for our database needs, our backend does not require external caching as Heroku Postgres already provides database caching.[1] Implementing an external cache will ruin the benefits of database caching.[2]
Layered system	Our backend makes use of a layered system. The Postgres database and the server-side code (NestJS) are run separately on Heroku. The clients make requests to the server via the API and the server gets the relevant data from Postgres. The client is unaware of the database implementation and cannot tell if there is any intermediary between the client and the end server.
Code on demand (optional)	This is an optional constraint and not required for a RESTful architecture. It is not implemented as static resources are sufficient for NUSCats needs.

[1] https://devcenter.heroku.com/articles/understanding-postgres-data-caching
[2] https://www.scylladb.com/2017/07/31/database-caches-not-good/

Milestone 6: SQL Queries

Share with us some queries (at least 3) in your application that require database access. Provide the actual SQL queries you use (if you are using an ORM, find out the underlying query and provide both the ORM query and the underlying SQL query). Explain what the query is supposed to be doing.

1. **GET** `/v1/sightings/latest`
   When a user first views the maps, the client needs the latest cat sightings for each unique cat to display them on the map. The client makes the GET request to the relevant endpoint and the backend will make the following query to the Postgres server:

   TypeORM query:

```
const queryBuilder: SelectQueryBuilder<CatSighting> =
  this.sightingsRepository
    .createQueryBuilder('sighting')
    .leftJoinAndSelect('sighting.cat', 'cat')
    .leftJoinAndSelect('sighting.owner', 'owner')
    .distinctOn(['sighting.cat_id'])
    .where('sighting.cat_id IS NOT NULL')
    .andWhere('sighting.type = :type', { type: SightingType.CatSighted })
    .orderBy('sighting.cat_id')
    .addOrderBy('sighting.created_at', 'DESC');

return from(queryBuilder.getMany());
```

   Underlying psql query:

```
SELECT DISTINCT ON ("sighting"."cat_id")
  "sighting"."id" AS "sighting_id",
  "sighting"."image" AS "sighting_image",
  "sighting"."cat_id" AS "sighting_cat_id",
  ST_AsGeoJSON ("sighting"."location")::json AS "sighting_location",
  "sighting"."location_name" AS "sighting_location_name",
  "sighting"."type" AS "sighting_type",
  "sighting"."is_seed" AS "sighting_is_seed",
  "sighting"."description" AS "sighting_description",
  "sighting"."owner_id" AS "sighting_owner_id",
  "sighting"."created_at" AS "sighting_created_at",
  "sighting"."updated_at" AS "sighting_updated_at",
  "cat"."id" AS "cat_id",
  "cat"."name" AS "cat_name",
  "cat"."image" AS "cat_image",
  "cat"."neutered" AS "cat_neutered",
```

```
  "cat"."one_liner" AS "cat_one_liner",
  "cat"."description" AS "cat_description",
  "cat"."zone" AS "cat_zone",
  "cat"."is_seed" AS "cat_is_seed",
  "cat"."created_at" AS "cat_created_at",
  "cat"."updated_at" AS "cat_updated_at",
  "owner"."id" AS "owner_id",
  "owner"."uuid" AS "owner_uuid",
  "owner"."username" AS "owner_username",
  "owner"."first_name" AS "owner_first_name",
  "owner"."last_name" AS "owner_last_name",
  "owner"."is_profile_setup" AS "owner_is_profile_setup",
  "owner"."profile_pic" AS "owner_profile_pic",
  "owner"."created_at" AS "owner_created_at",
  "owner"."updated_at" AS "owner_updated_at"
FROM
  "cat_sighting" "sighting"
  LEFT JOIN "cat" "cat" ON "cat"."id" = "sighting"."cat_id"
  LEFT JOIN "profile" "owner" ON "owner"."uuid" = "sighting"."owner_id"
WHERE
  "sighting"."cat_id" IS NOT NULL
  AND "sighting"."type" = $1
ORDER BY
  "sighting"."cat_id" ASC,
  "sighting"."created_at" DESC
```

Explanation:

The query selects all the cat sightings that have a cat tagged to it and with a sighting type of ‘CatSighted’. Order the sightings by cat_id then chronological order and get all the first sightings with a unique cat_id.
This way, we will get the latest cat sighting of each known cat.

2. **GET** /v1/sightings?ownerIds={id}&limit={12}&page={1}

When a user views his profile, we fetch a list of cat sightings that are made by him to display on his profile page. The client makes the GET request to the relevant endpoint and the backend will make the following query to the Postgres server:

TypeORM query:

```
let queryBuilder: SelectQueryBuilder<CatSighting> =
  this.sightingsRepository.createQueryBuilder('sighting');

queryBuilder = ownerIds
  ? queryBuilder.andWhere('sighting.owner_id = ANY(:ownerIds)', {
      ownerIds,
    })
  : queryBuilder;

queryBuilder = queryBuilder.orderBy({ 'sighting.created_at': 'ASC' });

return from(paginate(queryBuilder, pagingOptions));
```

Underlying psql query:

```
SELECT
  "sighting"."id" AS "sighting_id",
  "sighting"."image" AS "sighting_image",
  "sighting"."cat_id" AS "sighting_cat_id",
  ST_AsGeoJSON ("sighting"."location")::json AS "sighting_location",
  "sighting"."location_name" AS "sighting_location_name",
  "sighting"."type" AS "sighting_type",
  "sighting"."is_seed" AS "sighting_is_seed",
  "sighting"."description" AS "sighting_description",
  "sighting"."owner_id" AS "sighting_owner_id",
  "sighting"."created_at" AS "sighting_created_at",
  "sighting"."updated_at" AS "sighting_updated_at"
FROM
  "cat_sighting" "sighting"
WHERE
  "sighting"."owner_id" = ANY ({id})
ORDER BY
  "sighting"."created_at" ASC
LIMIT 12
```

Explanation:

Select all sighting information from the cat_sightings table where owner_id matches any ids in the array. Order the query result by time created in ascending order and limit the query response to 12.

Note: The pagination module builds the pagination filter (LIMIT+OFFSET) depending on the page number and page size of the query

3. **PUT** /v1/users/{uuid}
   When a user updates his profile information such as his first name, last name and profile picture, the client makes the PUT request to the relevant endpoint and the backend will make the following query to the Postgres server:

```
return this.profileRepository.update(
  { uuid: profile.uuid },
  {
    ...updateProfileDto,
    is_profile_setup: true,
  },
);
```

Underlying psql query:

```
UPDATE
  "profile"
SET
  "first_name" = $1,
  "last_name" = $2,
  "profile_pic" = $3,
  "is_profile_setup" = TRUE
WHERE
  "uuid" = $3
```

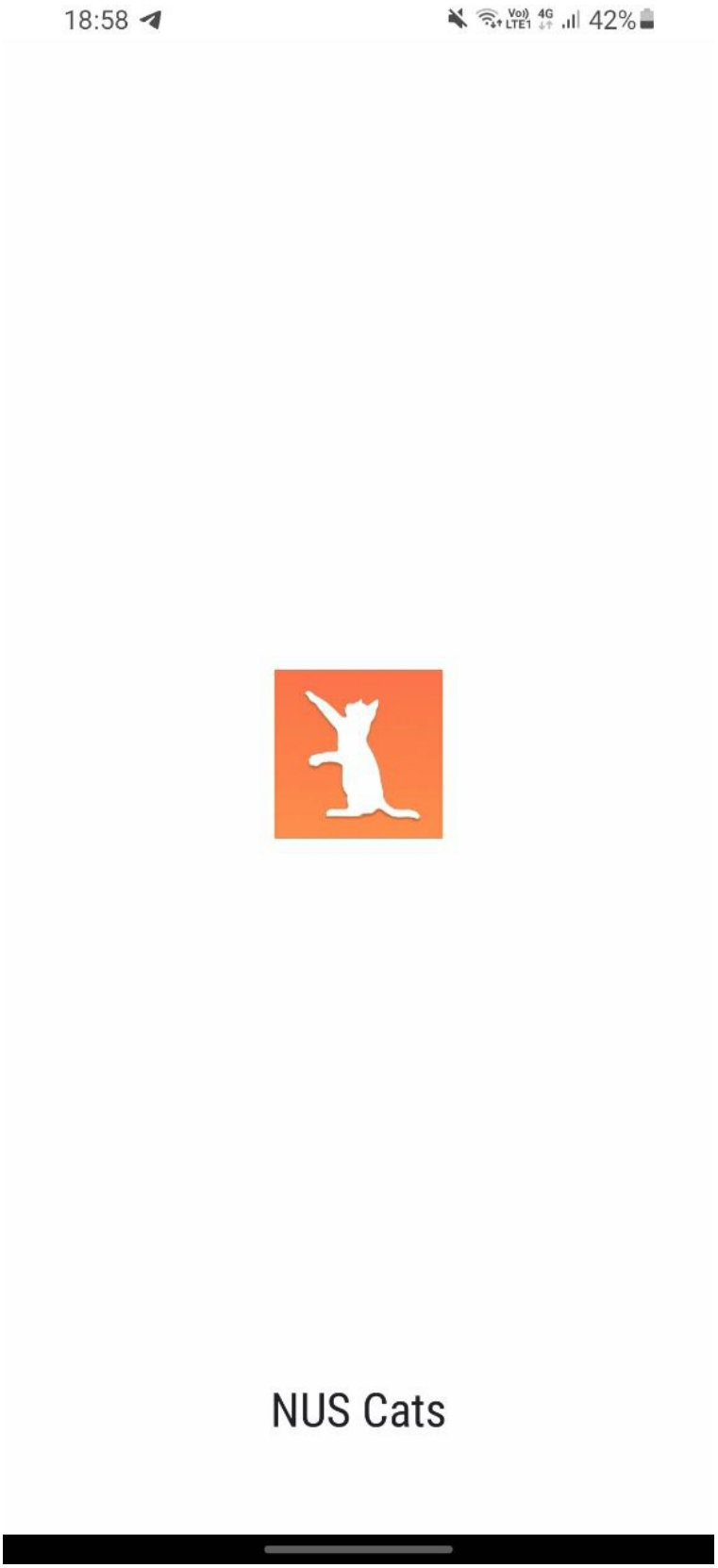
Explanation:
Update the provided profile details and set the is_profile_setup boolean to true in the profile table for entries that have the provided uuid. This will update the profile information of the specified profile in the profile table.

Milestone 7: Icon & Splash Screen

Create an attractive icon and splash screen for your application. Try adding your application to the home screen to make sure that they are working properly. Include an image of the icon and a screenshot of the splash screen in your write-up. If you did not implement a splash screen, justify your decision with a short paragraph. Add your application to the home screen to make sure that they are working properly. Make sure at least Safari on iOS and Chrome on Android are supported.

App Logo

Android Splash Screen

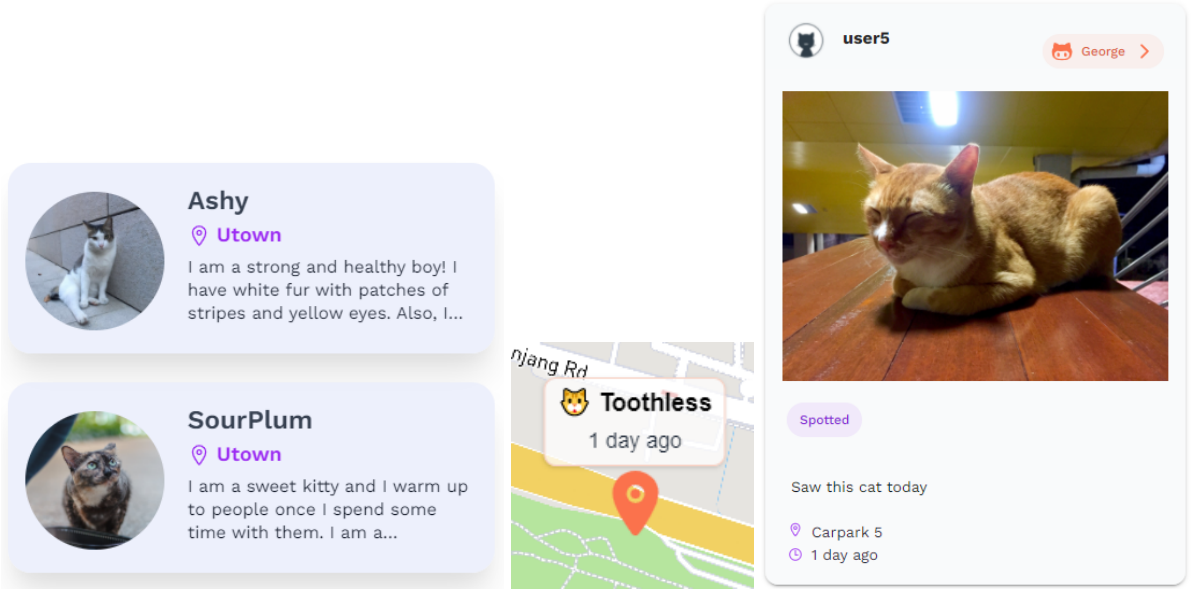


iOS Splash Screen

Milestone 8: UI Design & CSS Methodologies

Style different UI components within your application using CSS in a structured way. Explain why your UI design is the best possible UI for your application. Choose one of the CSS methodologies and implement it in your application. Justify your choice of methodology.

Our application's goal is to create a healthy and enjoyable experience, centered around providing welfare for NUS's cats. Thus, our ideal UI design should present an approachable and friendly feel for our users. To achieve this, we emphasized rounded corners, soft colours, and fewer borders.



For the choice of our CSS methodology, we decided on a utility-first CSS approach, using the Tailwind CSS framework. This is an innovative new methodology that has picked up major steam in recent years. The idea is to have a pre-created set of CSS utility classes like **rounded-sm** (rounded small) or **mt-5** (5 "units" of margin). These utility classes tend to do only one thing, making them reusable and composable. As a result, you can style any HTML by combining the utility classes, like so:

```
<div className="flex flex-col justify-start flex-auto mt-3 mb-4 ml-4">
  <p className="text-lg font-semibold text-gray-700">{cat.name}</p>
```

This was a great fit for our team's purposes for a few reasons:

First, we did not have a CSS expert on our team. Tailwind CSS helped by providing utility classes with great documentation for almost any scenario. This allowed us to add complicated effects like our own custom animations with minimal effort. Also, our application was mobile-first, but we also wanted a decent experience on desktops. This sort of responsive design is difficult to achieve normally. Our team could however leverage Tailwind CSS's built-in breakpoints to style our UI in a responsive manner easily.

Next, our team also did not have much design experience. Despite agreeing on a UI design, we were afraid our UI would end up looking inconsistent. Tailwind CSS solved this out-of-the-box, as using its fixed set of utility classes means that you constrain your team to a fixed design system. We could ensure that our application's font sizes, colours, overall look was mostly the same, by simply using the provided CSS classes. Despite following a constrained design system, Tailwind CSS still allowed us to customize it via configuration. This allowed us to bring in our own color scheme that we decided upon.

Finally, the biggest difficulty of the project was the limited timeframe, and thus, the speed of iteration was critical. A utility-first CSS approach helped with this tremendously. Other CSS methodologies would have required us to name every single class, and naming is one of the hardest problems in Computer Science. Using a utility first approach allowed us to completely skip that step. Also, co-locating styles with markup allowed us to write and edit all design-related components in a single place. This reduced the frequency of context switching between files.

Some critics of the utility-first methodology claim that it leads to messy and duplicated code. But, since we were using React, we could easily extract any duplicated code out into individual components. In this manner, we reduced duplication on the component level, instead of the CSS class level.

Another criticism is that the utility-first CSS methodology results in code that is lacking in readability due to the sometimes long class names. We felt this was a worthy trade off, as we felt the methodology also helped to make our CSS more maintainable. We did not have to worry about the complex concept of CSS specificity as Tailwind CSS's classes only affect the element you apply it to. This allowed us to add and edit our CSS without causing unintended side effects, avoiding the trap of the "append-only" stylesheets.

Milestone 9: HTTPS

Set up HTTPS for your application, and also redirect users to the https:// version if the user tries to access your site via http://. HTTPS doesn't automatically make your end-to-end communication secure. List 3 best practices for adopting HTTPS for your application.

Since we deployed with Heroku, users are able to access our site through Heroku via HTTPS which then proxies the requests via http internally to our application. However, due to the way Heroku forwards the requests onto our server, we used a third party library “heroku-ssl-redirect” that provides a middleware for this purpose. It checks for the headers set by Heroku when forwarding requests (“x-forwarded-proto”) and redirects the user if that particular header is not set to “https”.

1. Get your certificate from a reliable certificate authority
    a. Our application is deployed with Heroku, which provides free dynos with the certificate on the “*.herokuapp.com” domain. Heroku certificates are issued by Amazon which is recognized by many browsers, instilling a sense of security in our users.
2. Support HTTP Strict Transport Security (HSTS)
    a. HSTS tells browsers to request HTTPS pages automatically even if users request for http in the browser location bar and minimizes risk for users.
    b. Our application serves our PWA statically from the backend server. HSTS is implemented through the use of the “helmet” middleware that attaches HSTS headers to our responses.
3. Expired Certificates
    a. Our certificates are supplied by Heroku and hence, we are not at risk of certificates expiring and exposing users to malicious attacks.

Milestone 10: Offline functionalities

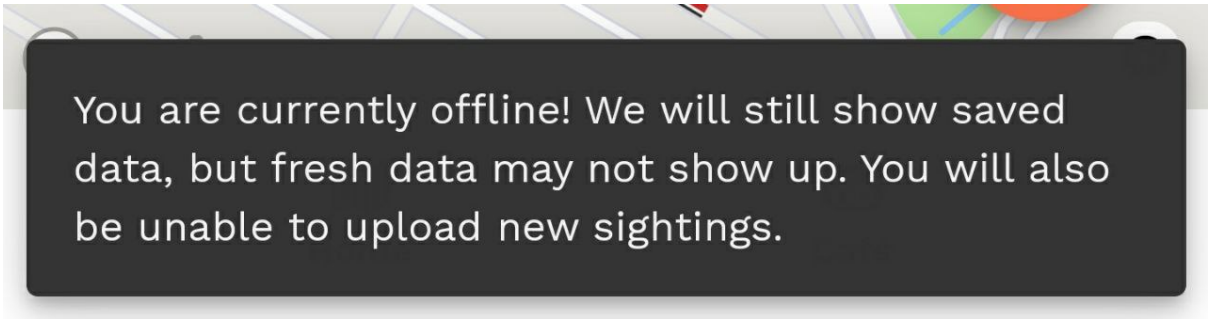
Implement and briefly describe the offline functionality of your application. Explain why the offline functionality of your application fits users' expectations. Implement and explain how you will keep your client synchronised with the server if your application is being used offline. Elaborate on the cases you have taken into consideration and how they will be handled.

To support offline functionality, our application uses service workers to cache different assets and resources. Some important areas of our application where we focused on offline functionality include:

Sightings

We cache the latest sightings of cats via a network-first strategy. Our application will first request for the latest sightings via the network. If successful, it will display those sightings and cache the results. Otherwise, the application will fall back to using the service worker cache. The network-first strategy is important, as up-to-date sightings are very important to our users. They need to know of any new sightings so that they have a better idea of where a cat is.

If a user is offline, they may still expect to view sightings on the map. For example, NUS Cat Cafe members may be out trying to feed the cats, and lose their internet connection. In that scenario, it would be useful to still display the last-known latest sightings of the cats. To ensure that users don't mistake the cached results for being the absolute latest results, we included a toast to notify the user when they are offline.



Also, the map in our application is an integral part of viewing sightings. Thus, we also cached the assets from Mapbox, the map provider we use, to ensure that users can view cached parts of the map while offline.

On the other hand, we decided not to enable uploading new sightings while offline. We felt a simple workaround for users would be to take a picture of the cat first, then to upload the image the moment the Internet connection is restored. Since NUS has Wifi coverage in most areas, users should likely be able to upload the pictures relatively quickly. Thus, we felt it was not a high priority to include offline support for this functionality

Cats

We cache the data of cats that the users view via a stale-while-revalidate strategy. This involves our application's service worker responding to requests for cats immediately from the cache. If the data is not cached, it falls back to the network. Regardless of whether the data is cached or not, the service worker still makes a network request to update the data in the cache. This allows us to respond to requests quickly to keep user interaction snappy. At the same time, we ensure that the latest versions are available on the next request.

This is a good fit for viewing cats - users will want to view cats and their profiles quickly. Also, having the latest data of a cat is not essential as the profiles are likely edited only rarely.

The admin functionality related to cats is also not available offline. Most of that admin functionality is done infrequently, and is not time sensitive. Thus, we similarly felt it was not a high priority to include offline support for this functionality.

Milestone 11: Choice of Authentication Scheme

Compare the advantages and disadvantages of token-based authentication against session-based authentication. Justify why your choice of authentication scheme is the best for your application

	Token-based Authentication	**Session-based Authentication**
Stateless	Information is stored in a token and signed by the server. The token is sent to the user and stored on the client-side. When the client makes a request, it sends the token to the server. The token and its stored information is sufficient to validate the user.	User details are stored in the session on the server side and a session identifier is sent to the user. This server uses the session to maintain user state such as whether the user is authenticated or not. This violates the stateless constraint of REST.
Server Management	The server only needs to verify the token signature to validate the user and the server can then use the token's stored information to process the user request. (eg. authenticated or not)	The server needs a way to store the session data in memory or database. The server also needs to efficiently look up user's information based on the provided session id
Scalability	There is no issue with scaling on the server side as token is stored client-side	Based on the management mentioned above, as the app scales in size, the size of the memory cache or database increases proportionally. This also affects the performance of the server
Size	Tokens do not have a limitation on the size of the data, but it is larger compared to session id. Clients have to attach the token for every	Session identifier does not contain any additional information aside from its id and is hence very small as compared to tokens.

	authenticated request which may result in a latency issue.	

Implementation

As there are many benefits of a token-based authentication over a session-based one, we decided to use token-based authentication for our application. We use JWT token format for our token and store the user identifier in the token payload. We attach this JWT access token into the client's cookie together with a JWT refresh token. The JWT access token is used for authenticating the user and has a short lifespan of 20 minutes, the JWT refresh token has a longer lifespan of 1 week.

Justification

The short life span of the access token helps prevent potential misuse of the token. The refresh token's purpose is to validate the user and provide a new access token. We store the refresh token in the database for validation and lookup. This workflow is more scalable than a session-based authentication as actual database lookup happens only when the user requires a new access token. Furthermore, the 20 minute limit is much greater than the expected usage of an average user and hence database lookup happens only occasionally.

To protect the cookies from cross-site request forgery (CSRF) attacks, we ensure that the cookie has a SameSite=Strict flag[3]. We also secure the cookie with a Httponly and secure flag to protect it against cross-site scripting (XSS) attack[4].

[3] https://portswigger.net/web-security/csrf/samesite-cookies
[4] https://resources.infosecinstitute.com/topic/securing-cookies-httponly-secure-flags/

Milestone 12: Choice of Framework & Mobile site Design Principles

Justify your choice of framework/library by comparing it against others. Explain why the one you have chosen best fulfils your needs. Lastly, list down some (at least 5) of the mobile site design principles and which pages/screens demonstrate them.

Choice of Framework

We had a few important criteria that affected our choice of framework. First, given the focus on a good mobile experience, we wanted a fully-fledged PWA framework. This would help us handle things like native page transitions with lesser effort. This ruled out UI frameworks like Vuetify and Material UI. While native page transitions using those frameworks were possible, setting them up would have taken more time compared to a PWA framework.

Another important criteria was community support. Given our lack of experience in creating good PWAs, we felt like we would rely heavily on the community for help. With that, we decided to focus on the 2 most popular PWA frameworks, Ionic and Framework7. Both had active contributions on GitHub and popular forums where users could ask questions.

Finally, another crucial criteria was the support for geolocation and camera functionality, as these 2 native features are the centerpiece of our application. For these features, Ionic documentation was stellar. Both features were easily available in the framework, with excellent documentation. Taking a photo was actually the example in Ionic's getting-started guide. Framework7 also supported those features. However, there was no clear documentation on how to get started with those features. Instead, we had to learn another tool, Cordova, which Framework7 was built on, to implement those features. Additionally, Ionic's community seemed much more active than Framework7's. Thus, it was an obvious choice to pick Ionic for our application.

Mobile site design principles

Many factors go into designing a good mobile site. Based on our research,[5] we picked out some design principles that we think would be most suitable for our application, and adhered to them as far as possible:

Create a solid hierarchy of information

The rationale for creating a solid hierarchy of information is to convey information to the user as quickly as possible. If done well, information of the functionalities in the page will be apparently to the user simply by scanning the page. There are a few sub-points to achieve this, namely:

- Keep important information and key call-to-action buttons front and center
- Visually separate navigation from content

As an example, this is the home page of our application, which is the very first page that the user sees after he/she logs in. Keeping in mind that the main functionalities of the application are (1) locating cats, and (2) creating sightings of

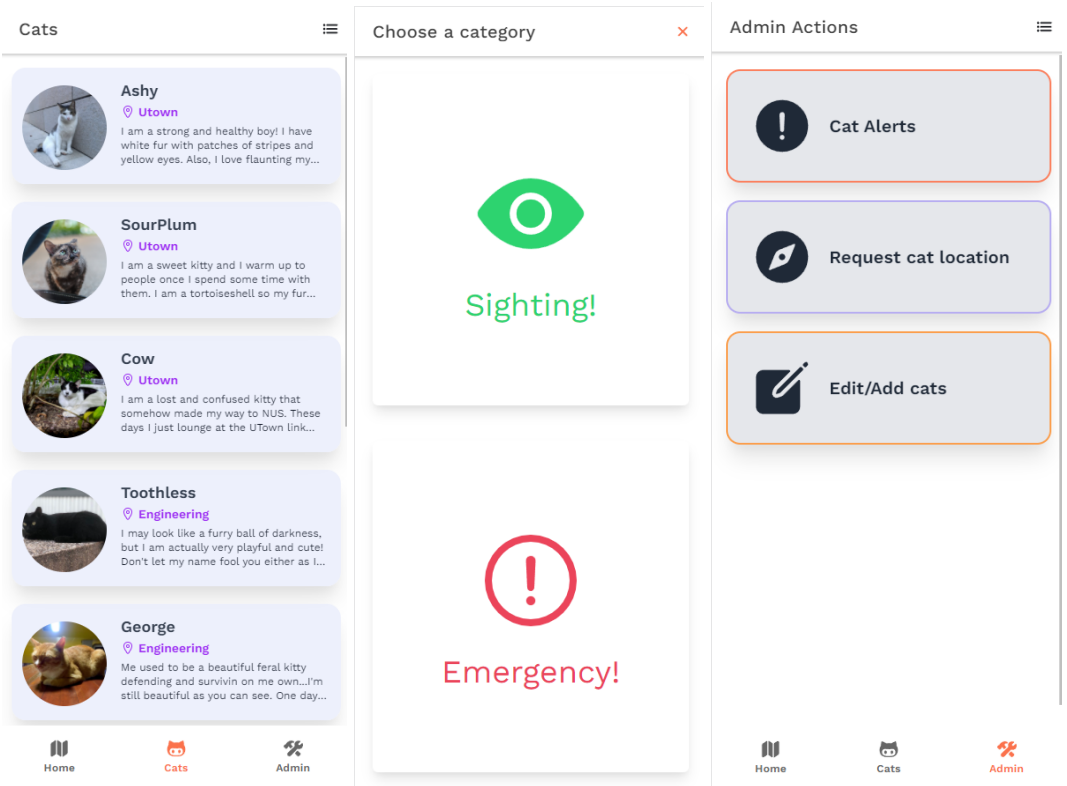
5 https://xd.adobe.com/ideas/process/ui-design/what-is-mobile-web-design/; https://uxplanet.org/principles-of-mobile-site-design-c4c721693c42

cats, we made sure that these two functionalities are available on the home page, at the forefront of our entire app.

We note that the camera button to create a new sighting is relatively small, at the side of the screen, but this is a carefully considered design trade-off. If we want the map to be available on the home page, and be as big as possible (for ease of use), then the best way to present the create sighting function would be as a floating button on the map itself. We concluded that this strikes a good balance of displaying both functionalities on the home screen.

We also made sure that the content of the app is clearly separated from navigation. Navigation options are found either at the bottom or at the top of the screen, and are clearly demarcated from the contents of the app.

Use finger-friendly touch targets



We are all too familiar with the frustration of trying to click buttons that are too small, or accidentally clicking buttons we did not intend to click because our fingers are too fat. We therefore made sure that clickable items are made as large as possible. The above three pages are examples of our deliberate attempts at making items large enough that even King Kong would have no problem tapping on them.

Let users explore before they commit

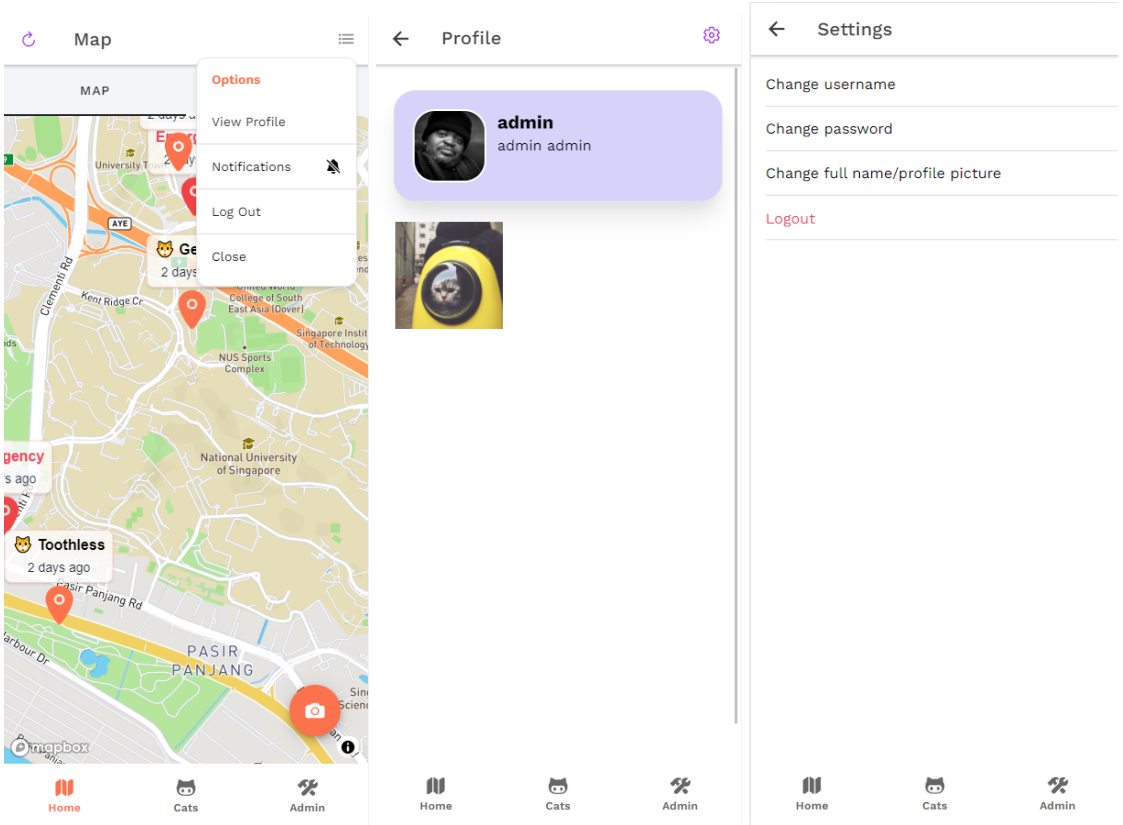
Requiring users to sign up before they are able to use an app can drive potential users away. Some users might be unwilling to go through the hassle of signing up for an account if they are not sure if they will use the app subsequently. Thus, we allow users to “continue as guest”, which allows them to explore the app before signing up.

Limit the number of columns

This seems like almost a cardinal rule of good mobile site design. Considering the usual shape of smart phones (ie. an elongated rectangle), there really isn’t much space for more than one column to be displayed. As such, we have kept all content in our application limited to one column.

Keep menus short and useful

Long menus can leave users confused, as it makes it difficult for them to find their desired button. As such, we made the decision to separate the “general” options from the options that are specific to the user profile.



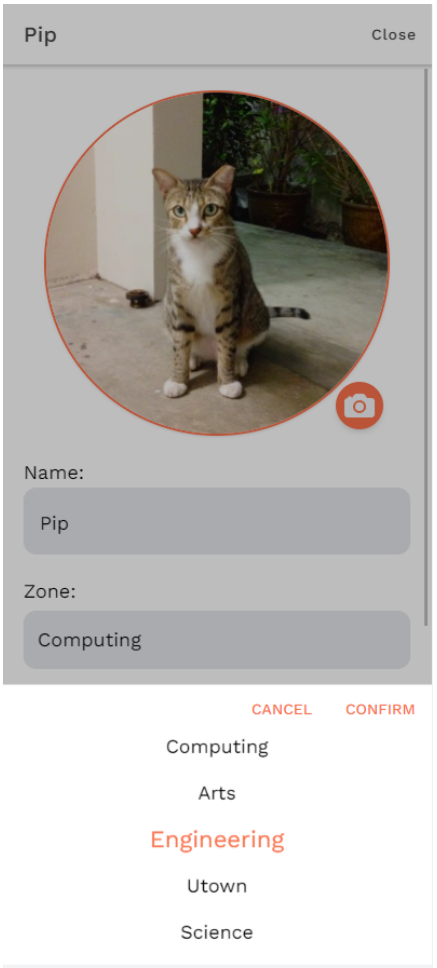
With references to the screenshots above, general settings are found under the hamburger icon on the far-left screen, whereas settings specific to the user profile is found at the profile page, in the cog icon at the top right-hand corner of the screen in the middle. This leads to the screen on the far right.

This design allowed us to keep both menus within 4 options, which reduces the risk of confusing the user.

Minimize user effort for data input

Lastly, we have made it a point to minimize user effort for data input by limiting the required input to only the essential ones. For example, when signing up, we only require users to provide us with their email, username, and password, which are the bare minimum to create a working account.

Additionally, when asking for input of fields that we know would be limited by a predefined set of options, we allow users to select from a list. For example, when selecting the school zone that a cat should be classified under, users would be presented with a pop-up option like such:



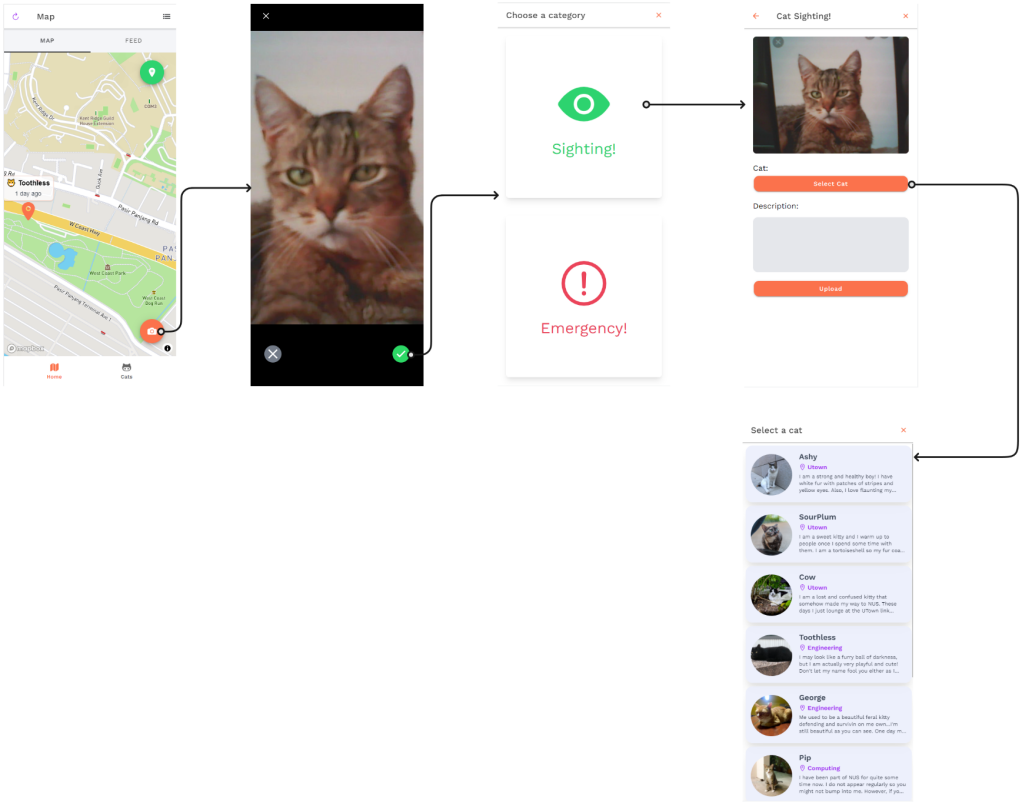
This allows them to easily select their desired option from the predefined list, instead of manually keying in the options. Doing this also eliminates the possibility of typo errors for that field.

Milestone 13: Main User Workflows

Describe 3 common workflows within your application. Explain why those workflows were chosen over alternatives with regards to improving the user's overall experience with your application.

Upload sightings

Uploading cat sightings is one of the main functionalities of our app. Thus, when designing the workflow of uploading sightings, we kept in mind that it had to be accessible and simple to use.



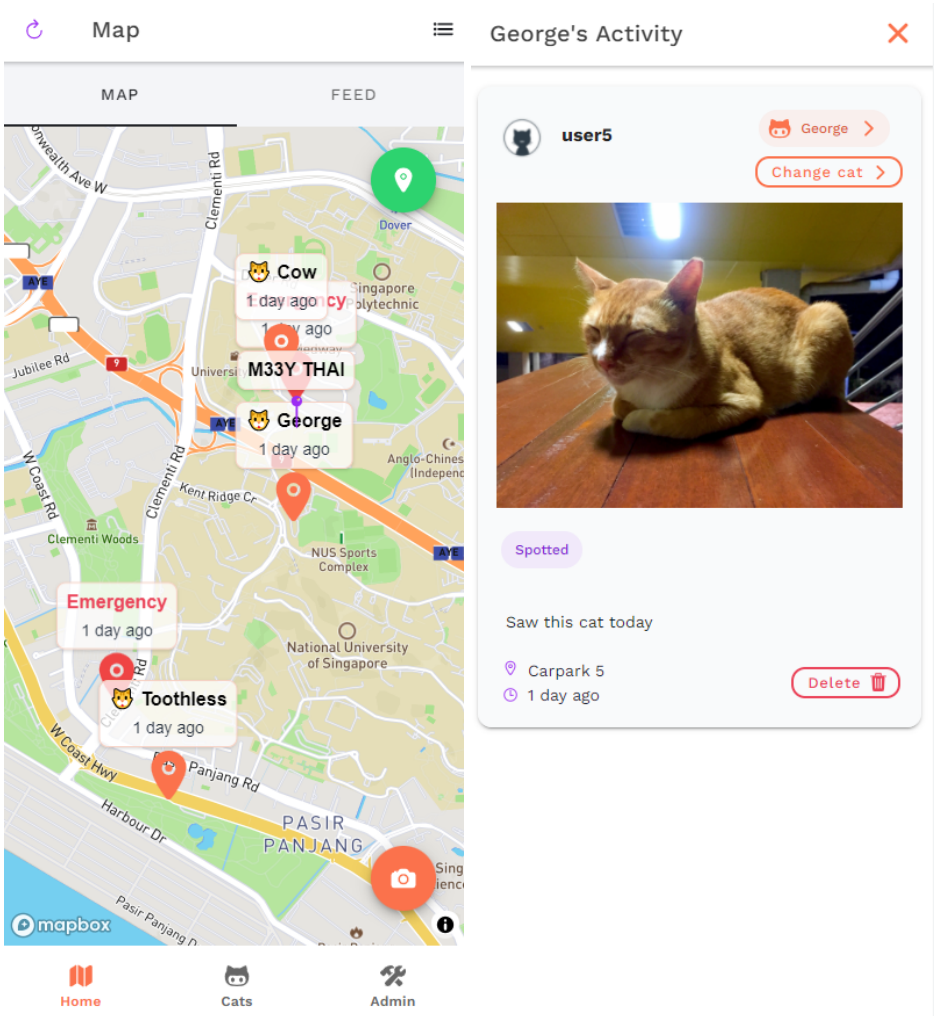
With reference to the user flow diagram above, if a user wants to bring up the camera to take a photo, he/she would tap the camera button on the home page. We deliberately placed the camera button on the home page as it is the most accessible location for the user. When the user opens the app, as long as he/she is already logged in, the camera button would immediately be available. There is therefore no need for the user to navigate through unnecessary flows just to reach the camera function. He/she can access the camera by a single tap after launching the app.

Next, after taking the photo, the user can select between the two categories, namely a normal “sighting” and an “emergency”. Here, appropriate colour-use is combined with universally understandable symbols to show that “emergency” should only be clicked in exceptional situations. Instead, for normal “sightings”, the user should click the more inviting green button.

The next page leads to a short form that allows the user to select the name of the cat that he/she just sighted, and also provide a short description of the sighting. We are aware that not everyone is familiar with the names of the cats in NUS. Thus, we made sure that clicking on the “Select cat” button would bring up a list consisting of the images of the cats accompanied with the relevant information that would help a user identify the cat.

In any case, it is worth noting that “admin” users (or NUS cat cafe members) can subsequently change the selected cat that is labelled for the sighting, if users wrongly identify a cat.

Find cats



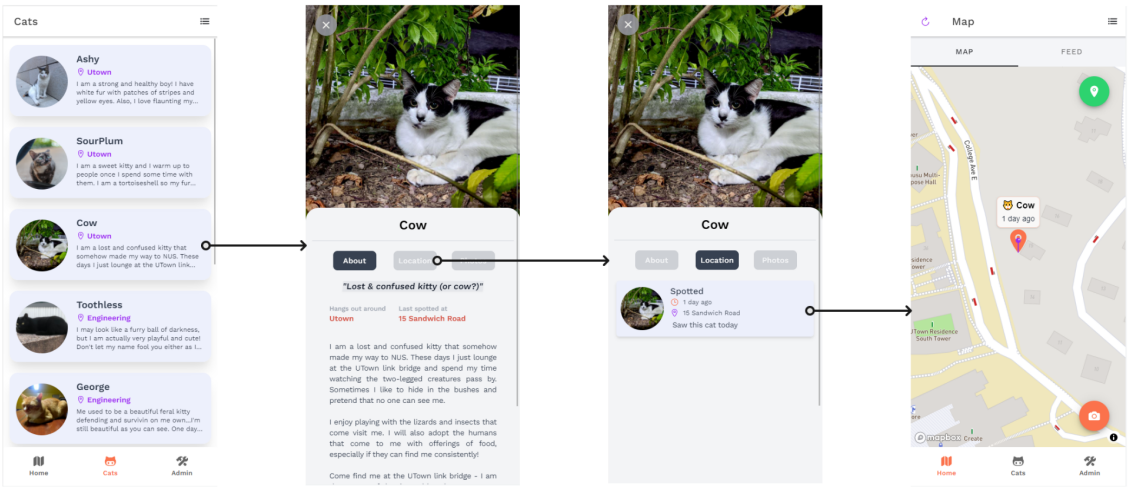
Alongside uploading cat sightings, locating cats is the main functionality of the application. As such, we placed it at the homepage of the application so that it is presented at the forefront.

The location of cats is represented as pins on a map, as shown above. This gives users a “birds-eye-view” of the location of all the NUS cats at a glance. From a user’s perspective, this would allow them to immediately see the cats that are in their immediate vicinity, which is helpful if they are looking to go cat-spotting. Emergency pins will also be displayed on the map, which allows NUS Cat Cafe to dispatch the nearest member, resulting in a timely response.

Details of the sighting can be viewed by tapping on the pins on the map. This brings the user to a page showing the latest sighting, as well as the history of sightings of that particular cat. While such information is also available on the “Feed” view, we felt that this would be more intuitive, since users can immediately access the details of the sighting from the map. The alternative would be for the

user to go to the “Feed” view and search for the relevant sighting of the cat, which is inconvenient and troublesome.

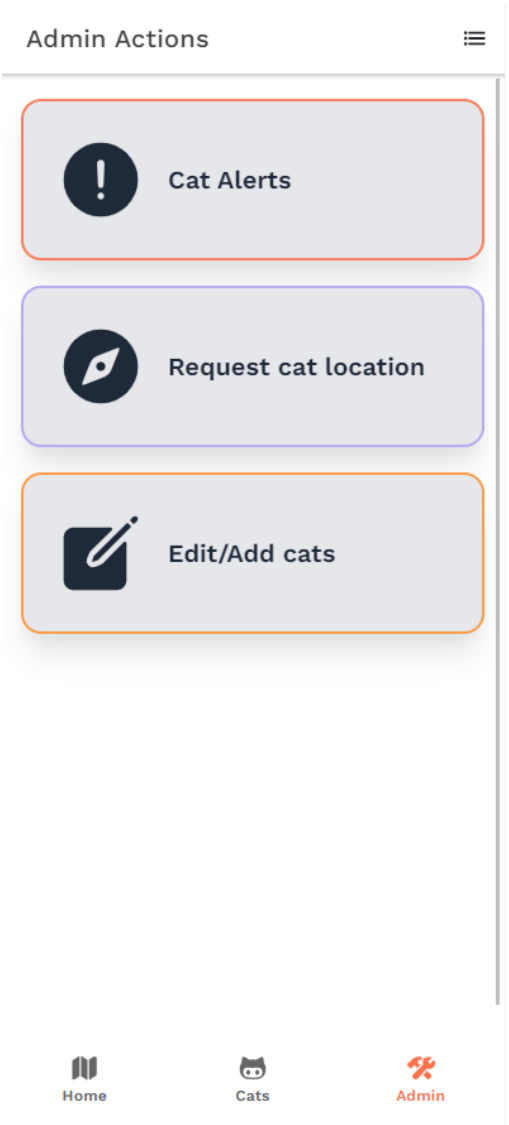
Alternatively, if a user wants to locate a particular cat (this use-case would probably be most useful for NUS Cat Cafe members who would like to feed a cat), he/she may follow the following user flow:



As seen on the above diagram, following the flow above would lead the user to the same map, centred on the selected cat. For locating a particular cat, we decided to make the function inside the cat’s information page as we felt that this functionality would be mostly used by Cat Cafe members as opposed to general users. There is, therefore, no need to bring this functionality to the forefront.

We are aware that this means that every time a member needs to focus on a cat to locate a cat, he/she has to go through the flow above, which can arguably be troublesome. Nevertheless, this is a trade-off we had to make. In a bid to make the UI of the main pages as clean as possible, we decided not to put this functionality at the forefront.

Admin flow



NUS Cat Cafe members have an extra “Admin” tab available, which leads to the page shown above. Each of the buttons correspond to a particular admin functionality available to members. Specifically, the above buttons allow members to manage cat emergency alerts, request for the location of cats, and edit/add to the cats stored in the database respectively.

We initially considered a design where such admin functions were scattered throughout the app. For example, the button to request for a cat’s location would be found in the respective cats’ page. However, we decided against it as we found that it would be neater to place admin actions in one place.

Firstly, placing admin actions in one place would allow Cat Cafe members to easily find the admin functions. Scattering these functionalities all over the app might be less intuitive, and more effort may be required to find such functions.

This is especially so for some admin actions that might only be used once in a long while, such as the edit/add cats function.

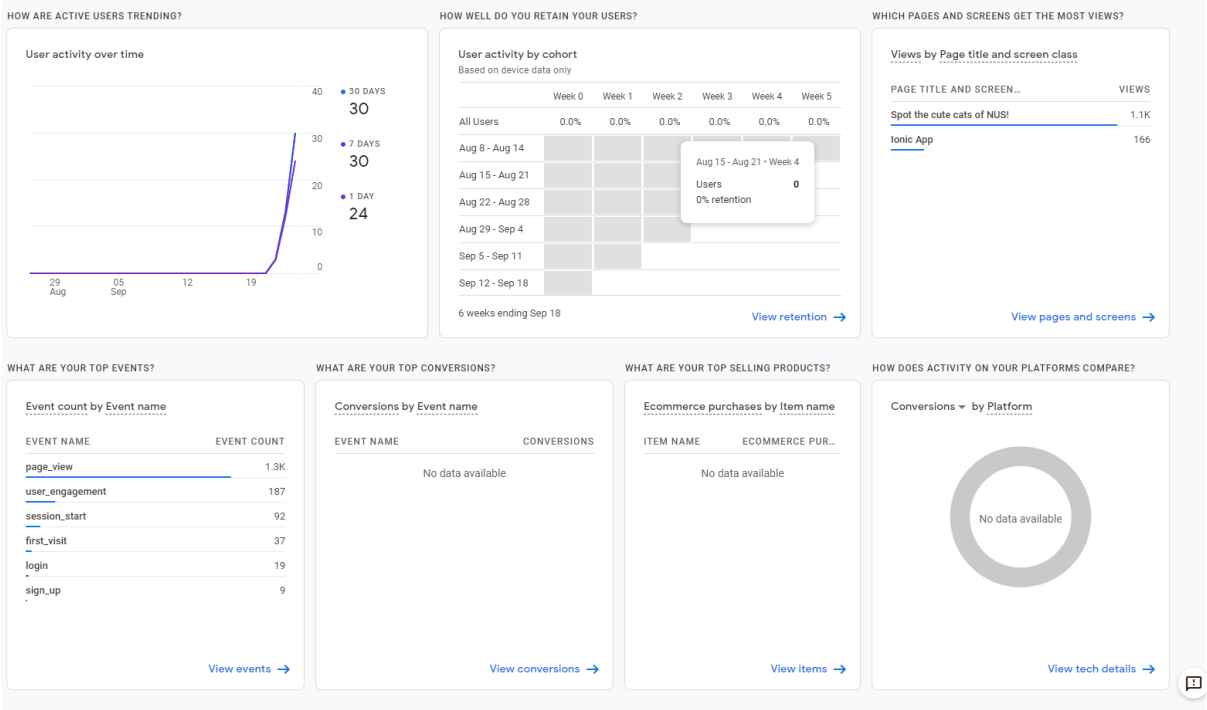
Secondly, it is conceptually neater to keep the admin portion of the app separate from the “normal” parts of the app. NUS Cat Cafe members might want to use the app as a normal user, and share their sightings like others do. Scattering admin actions throughout the app might interfere with the user experience, since the UI would necessarily be cluttered with admin buttons. Separating the admin functions into a separate tab has the effect of “partitioning” the app into two parts. Thus, when members want to use the app merely to upload sightings, they can access one part, and if they want to use it for its admin functionalities, they can use the other part of the app. Both parts are separate and do not interfere with each other.

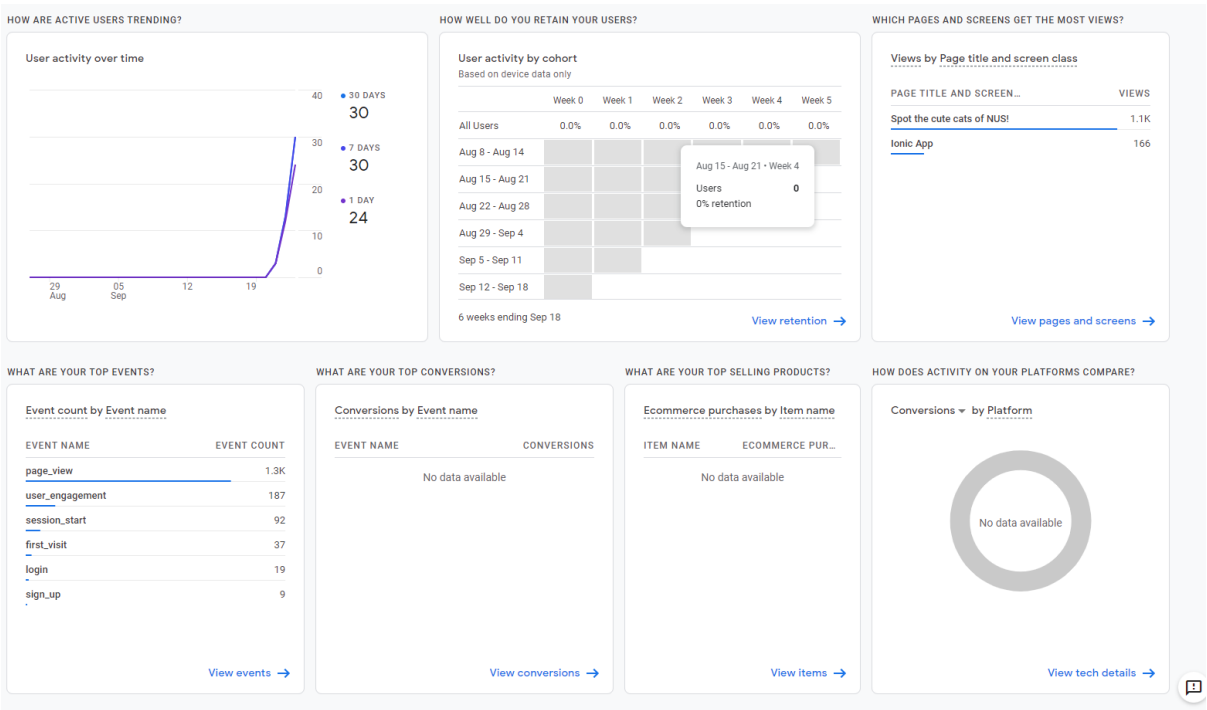
Milestone 14: Google Analytics

Embed Google Analytics or equivalent alternatives in your application and give us a screenshot of the report.

Get the report in pdf format here: google-analytics-nuscats-report.pdf

Report Screenshot:

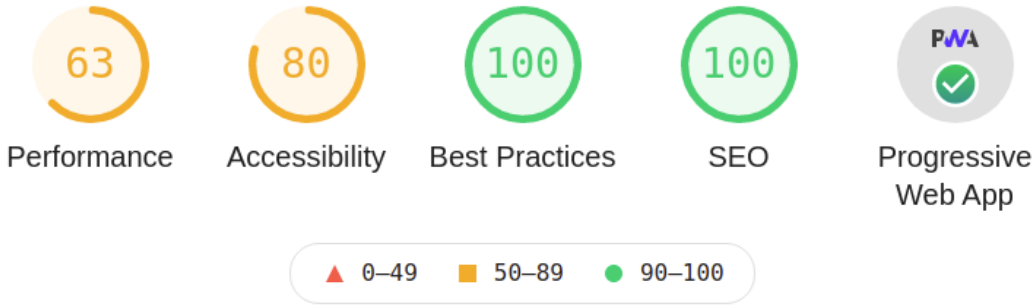




Milestone 15: Lighthouse Report

Achieve a score of at least 8/9 for the Progressive Web App category on mobile (automated checks only) and include the Lighthouse HTML report in your repository.

Our lighthouse report is located in the Github repository:



https://github.com/cs3216/2021-a3-mobile-cloud-2021-a3-group-11/blob/main/group-11-lighthouse.html

Milestone 16: Social Plugins

Identify and integrate with social network(s) containing users in your target audience. State the social plugins you have used. Explain your choice of social network(s) and plugins.

For our application, we have included social share buttons for Facebook and Telegram to allow users to share a cat sighting with others.

Since our information is sourced from the community and NUS Cat Cafe is a non-profit organization, it is important to be able to grow organically. Hence, we have decided that a share function would greatly benefit our application as cat lovers would be proud to show off the cat pictures they have taken and showcase their contribution to the cat community. NUS Cat Cafe members can also share these sightings individually rather than cluttering up their official page on Facebook and Instagram.

Facebook is an important medium as that is what NUS Cat Cafe uses to disseminate their information and the ability to share sightings with their community would be very useful. Telegram is also one of the most popular messaging platforms used by NUS Students and a place where they would be more likely to share information with one another. Hence, we have decided to target these two platforms for our share sightings function.

Milestone 17: Geolocation API

Make use of the Geolocation API in your application.

Geolocation is an integral part of our application. We use it in two ways.

Show nearby cats

For cat lovers in NUS, we use their location to locate them and center the map on their coordinates, allowing them to view the cats nearby and show their location relative to the cats. Users can navigate around the map and click a button to center the map back onto themselves. Navigation with the users' coordinates also makes it easier for Cat Cafe members to orientate themselves.

Cat sighting coordinates

We want to make it as simple as possible for users to report cat sightings. We achieve this by utilising their current location coordinates whenever they create a cat sighting. This reduces the need for users to manually input a location on the map whenever they wish to create a cat sighting. Users need not know which Computing Drive they are currently on in order to give Cat Cafe or other users the accurate location of the cat. This also serves as a light barrier to malicious acts where users attempt to give random locations of cat sightings.

Milestone 18: Coolness Points

Camera API

Our application also utilises another aspect of mobile devices which is the Camera API. This allows users to take live sightings of cats directly from our website rather than having to take a picture of the cat from their native application before uploading it on our site. By doing so, we hope to keep the entire user journey on our application, from opening up the application to look for cats, navigating to the cats and then snapping a photo of them to prove that you caught the cat!

Firebase Cloud Messaging

Another aspect of mobile devices that was implemented is the Push Notification feature. We achieve this by using Firebase Cloud Messaging.

For first time users of NUS Cats, the app will ask the user for push notification permission *(after the user has tried out the application and roughly knows what the app is about)*. The push notification is an opt-in feature that the user can enable. After enabling notifications, the admin can push out cat location request alerts to the user to get the community to actively search for the cat! *(on a voluntary basis of course)* This feature is useful for NUS CatCafe members, when the cat is nowhere to be found during feeding hours.