MothClassifier Design Documentation

Ben Giangrasso, Reese Jones, Abid Ahmed

Project Overview

Ecologists classify species to observe their role in their respective ecosystems. Discover Life is an ecological organization that classifies images of insects. However, their current process of image classification requires the certification of multiple experts, and the entire process takes around 35 seconds per image. Currently, Discover Life has thousands of unclassified images of moths. Image classification from an autonomous system will expedite the process and will remove the need of humans to classify images by hand. MothClassifier will classify the thousands of unclassified images from the Discover Life website using machine learning. These classifications will then be sent back to the Discover Life team. Additionally, MothClassifier will allow both researchers and non-ecologist users to receive instant moth species classifications through a mobile application interface. Users will be able to submit pictures from their camera or image gallery, and the application will provide a species name using the same machine learning model that will be used to populate Discover Life's website with classifications.

Users and Use Cases

The primary users, Researchers, are going to be biologists utilizing the mobile application to streamline the process of collecting and classifying moth data for the Discover Life project. Another class of user, a Non Researcher, are accounts for members of the general public to contribute images to the project and classify their own personal images. Each account type can have only one profile on the app, but as a result of the different 3rd party login methods supported can have multiple different login methods associated with the same user data. A Researcher's profile will be used to show which images need a manual reclassification based on poor confidence from our Machine Learning models. Both Researcher and Non Researcher accounts will have the ability to upload any photo from their photo gallery or the actual mobile device camera.

The mobile application will be available to any iOS user running iOS version 9.0 or later, and any version of android greater than or equal to SDK version 21 (Android 5.0, Lollipop). The methods for communicating with the machine learning models, the image database, user data store, and Google Cloud Functions are exposed with a RESTful interface and are not limited to any device. Currently the API is only accessible using the mobile app or through the Django REST Framework api browser exposed through the framework itself, which is only utilized for development.

Future Additions

The system is being designed in such a way that will allow for these additional features to potentially accommodate additional use cases by biologists at Discover Life.

- Creation of an Administrative panel on the Web so that Research teams can monitor data and statistics about image submissions over time.
- Addition of an option within the mobile application to store the most recent versions of the machine learning models on the device. This will allow for classification without an internet connection.

Design Overview

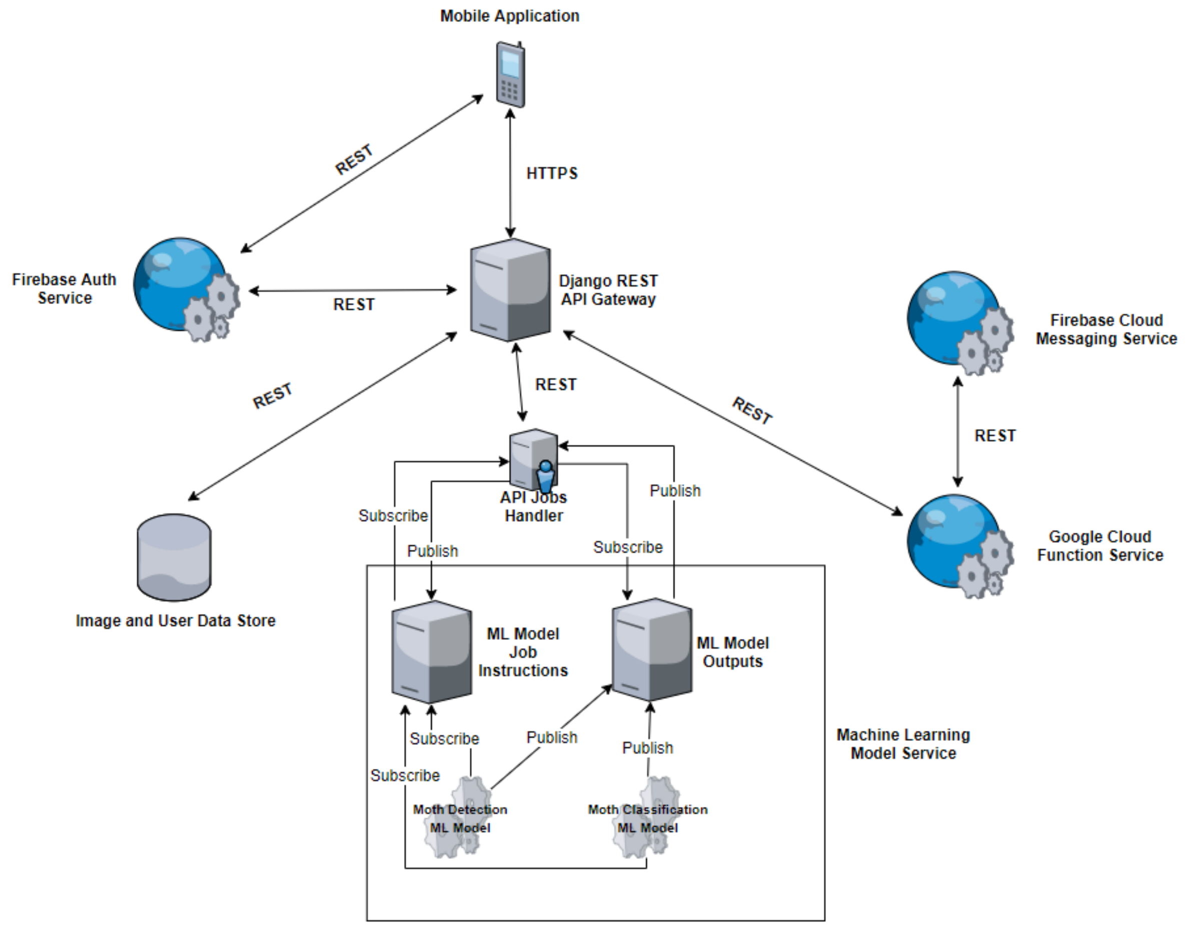


Illustration 1: Server-Side Design Architecture of MothClassifier project
(link to original image)

The moth classifier system is designed using a Microservices architecture for the server-side systems. The above illustration (Illustration 1) displays all of the main interactions between different services. In our design, the main Django REST API functions as a gateway and a coordinator between the different services utilized to provide functionality to the rest of the project. Communication between the mobile application and the Rest API which is facilitating interactions with services is done using RESTful calls protected by HTTPS. Each of the server-side components is designed to run on a VPS hosted on the Digitalocean cloud.

Generally users will submit image data (both classifications and submissions) through the mobile application, which will then be sent over to the API through REST calls. From there, the API will publish images, as well as the "jobs" associated with their classification, to the machine learning service. Within the machine learning service, the actual component running the machine learning tasks will subscribe to the job handler within the service, and its output will be consumed by the job handler. The job handler will distribute and consume all data relating to the actual classification and identification of the different machine learning models, as well as informing the ML service when to retrain and what information to retrain with. The only information which will be directly accessible to Researchers and general user accounts is the information which has been updated within the User and Image data store by the Django API and exposed through the mobile app interface.

Additionally, when the API gateway receives updated information regarding the conclusion of a classification or identification job, it will send out a REST call to a Google Cloud Function that is responsible for triggering Firebase Cloud Messaging service for asynchronous notifications. Another form of asynchronous communication used throughout the design are calls to the Firebase authentication service from both the mobile app directly and through the api gateway to enforce security.

As indicated by Illustration 1, all components will interface their requests through the REST api, which will be responsible for coordinating jobs to the Machine Learning service, the Google Cloud Functions, and the Firebase Authentication service. The machine learning service will interface with the API's job handler using a Pub/Sub pattern facilitated by the Redis message broker which will sit between the two services.

The actual data storage for the project is a PostgreSQL database which will be managed by the Django REST API. All of the image data will be stored on the same VPS that the API is hosted on, and the database will contain links to each image which can be sent out to the machine learning service to facilitate classification and identification. Storage for authentication information for Firebase authentication is handled in the cloud by the service itself, so the format of that data is irrelevant to the rest of the project. Lastly, iterations of different machine learning models will be stored in h5 format on the VPS that houses the machine learning service architecture.

API and Database Architecture

The Moth Classifier API is built with the Django REST Framework and contains various endpoints to handle the functionality of the system. The Moth Classifier API is seen as the glue that holds the entire Moth Classification system together. Thus, it is responsible for the business logic of the system. To do that, the API is broken up into different major endpoints that are responsible for executing requests. Each category of endpoints revolve around specific database models that store information vital to the entire system. The endpoints are as follows:

- users - Handles the logic relating to users of the Moth Classifier API
    - users/, users/{uid}/
- images - Handles the logic relating to the images stored in the Moth Classifier API
    - images/, images/download/, images/{id}/, users/{uid}/images/
- jobs - Handles the logic relating to jobs performed by the Moth Classifier API
    - jobs/, jobs/{id}/, images/{id}/jobs/
- classifications - Handles the logic relating to the classifications in the Moth Classifier API
    - classifications/, images/{id}/classification/
- models - Handles the logic relating to the machine learning models utilized by the Moth Classifier API
    - models/, models/{id}/, jobs/{id}/models/

The Moth Classifier API will interact with anonymous users, standard users, researchers, and administrators, so there are specific access controls in place to secure the API, prevent data leakage, and unauthorized access.

The users endpoints revolve around the User model which contains information relating to a user such as his/her email, name, and user status. Through the users/ endpoint, administrators of the system can view a list of all users registered with the system. Through the users/{uid}/ endpoint, administrators can delete users from the system. In addition, users have the ability to view their own profile and can make changes to their profile such as changing their name. Anonymous users cannot make any requests to the users endpoint and will be blocked from doing so.

The images endpoints revolve around the Image model which contains information relating to an image such as the actual image, location metadata, associated jobs, and an associated classification. Through the images/ and images/{id}/ endpoint, anyone can view images in the system. Anyone can download a folder containing images to curate their own dataset by accessing the images/download/ endpoint. Users are able to submit an image to the system and view a list of images they submitted to the system by accessing the users/{uid}/images/ endpoint. Finally, through the images/{id}/ endpoint, administrators or the user that submitted the specified image can delete that image.

The jobs endpoints revolve around the Job model which contains information such as the type of job, the associated image, related machine learning models, and job status. Only administrators can view a list of all the jobs in the system through the jobs/ endpoint, but users can view the jobs they submitted for each of their images through the images/{id}/jobs/ and jobs/{id}/ endpoints. Administrators or users can submit jobs for a specific image using the previous images/{id}/jobs/ endpoint. Anonymous users are forbidden from making any requests to this endpoint.

Since handling jobs requires time as they involve communicating with and running machine learning models, jobs must be handled asynchronously to provide a responsive user experience. Thus, the system will utilize Celery to handle jobs asynchronously. Celery is a distributed task queue system that relegates tasks to worker threads that run in the background. Celery uses message passing as its form of communication. Thus, the Moth Classifier API will designate jobs as Celery tasks and Celery will relegate these jobs to worker threads. This lets the API return an HTTP response for a job right away (instead of waiting for the job to start running and complete) and can tell the user that the job has been issued. In the background, the worker thread will handle and complete the job. Once the job has been completed, Celery will let the API know and the API can then update the job status to finished and modify the necessary endpoints such as updating a classification.

The classifications endpoints revolve around the Classification model which contains information such as the associated image, the species of moth, and the accuracy of the classification. Anyone can view classifications in the system through the classifications/ or images/{id}/classification endpoints, but only researchers or administrators can make changes to a classification through the images/{id}/classification endpoint.

The models endpoints revolve around the MLModel (short for Machine Learning Model) model which contains information such as a name for the model, the type of model, a rating for the model, and miscellaneous comments. The endpoints are for administrative purposes, so only administrators have the authority to access these endpoints. Administrators can view all the models utilized by the system and can add models to the system through the models/ endpoint. They can update the information for existing models through the models/{id}/ endpoint. Finally, administrators can view the models for each job through the jobs/{id}/models/ endpoint.

API Communication with Mobile Application

Generally, the communication between the REST API and the mobile application will be focused around four main functions: collecting/distributing user data, submitting image information, submitting updated classification information, and verifying authentication information.

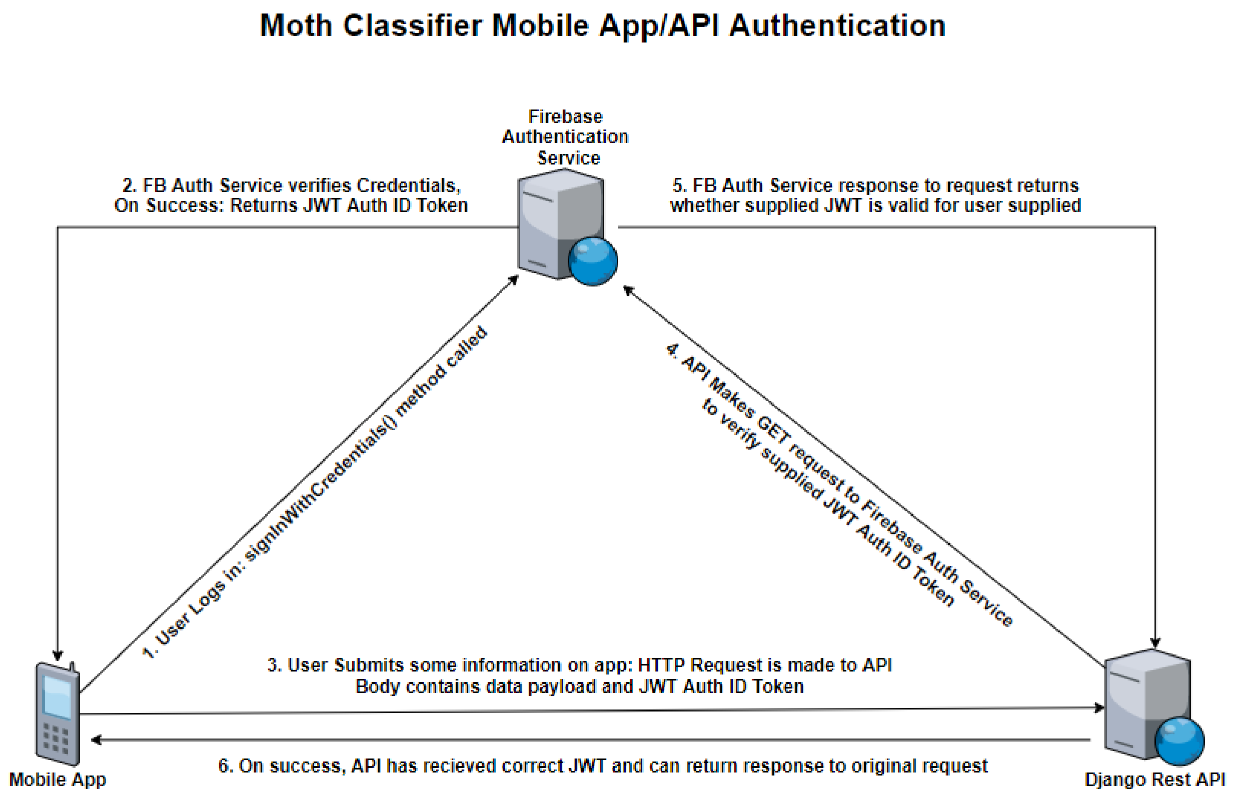
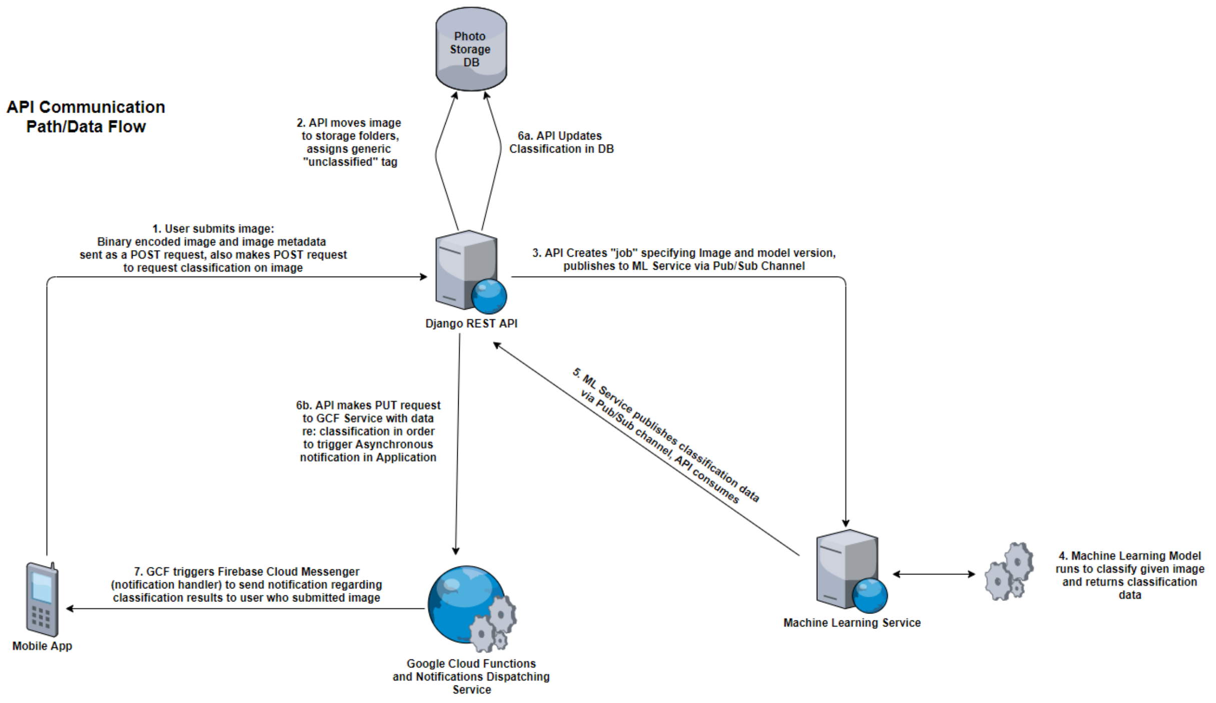

Moth Classifier Mobile App/API Authentication
Firebase Authentication Service
2. FB Auth Service verifies Credentials, On Success: Returns JWT Auth ID Token
5. FB Auth Service response to request returns whether supplied JWT is valid for user supplied
1. User Logs in: signInWithCredentials() method called
4. API Makes GET request to Firebase Auth Service to verify supplied JWT Auth ID Token
3. User Submits some information on app: HTTP Request is made to API Body contains data payload and JWT Auth ID Token
6. On success, API has recieved correct JWT and can return response to original request
Mobile App
Django Rest API

Illustration 2: Mobile Application and REST API Authentication process
(link to original image)

Authentication between the REST API and the mobile application is facilitated by an exchange of information between the two services as well as the Firebase Authentication service as shown in the above diagram (Illustration 2). To start, when a user logs into the application, a backend call is made (signInWithCredential()) to the Firebase Authentication Service, which will return a JSON Web Token (JWT) authentication ID token. This is the unique identifier for the account, and that key will then be stored in the mobile application and refreshed as needed. When a user submits some information or does something which will trigger a call to the API, the HTTP request will be made, with the header containing the JWT auth ID token provided by the Firebase Service. Upon receipt of a request, the API will take the supplied auth token and make an HTTP request to Firebase in order to verify the token is active, correct, and corresponds to the user account making the request. Upon verifying that information, the Firebase Service will respond to the API request with a confirmation of correct information. Only after that exchange is made and the confirmation is positive will the API fulfill any request made by the mobile application. The remaining three types of interaction between the mobile application and the REST API will be preceded by the authentication scheme described above, and each of the following API communications are reflected in the diagram below (Illustration 3).



API Communication Path/Data Flow
Photo Storage DB
2. API moves image to storage folders, assigns generic "unclassified" tag
6a. API Updates Classification in DB
1. User submits image: Binary encoded image and image metadata sent as a POST request, also makes POST request to request classification on image
3. API Creates "job" specifying Image and model version, publishes to ML Service via Pub/Sub Channel
Django REST API
5. ML Service publishes classification data via Pub/Sub channel, API consumes
6b. API makes PUT request to GCF Service with data re: classification in order to trigger Asynchronous notification in Application
4. Machine Learning Model runs to classify given image and returns classification data
Machine Learning Service
7. GCF triggers Firebase Cloud Messenger (notification handler) to send notification regarding classification results to user who submitted image
Mobile App
Google Cloud Functions and Notifications Dispatching Service

Illustration 3: API communication with Mobile App/Data Flow
(link to original image)

For the collection and distribution of user data, the mobile application will be responsible for collecting personal information about users at registration (or at time of first login if the user is logging into the application using a third party service like Google sign in or Sign in with Apple), and then will make POST requests to the /users/ endpoint of the API to update their information within the database. At the time of each user's login to the application, the app will make a GET request to the /users/{uid} endpoint of the API in order to get back all of the information relevant to a user's account. This data will then be stored in a Stream Provider at the top level of the application such that the API call need only be made once, and then any component descending into the widget tree can access that information as if it is a discrete value. This is highly performant for our purposes, as API requests need only be made at the time of login or at the time that a user updates their personal info.

For the submission of image information to the API, the application will rely on an endpoint for image creation. As the app will already have stored the authentication web token and the image file (as a user cannot submit an image to the app without an authenticated login), the request to the API endpoint will contain: the image encoded as a binary string, the image metadata (scraped at the time of submission), and the user's identifying information (user id, email, and auth token).

Similarly to submission of image information and metadata, the submission of updated classification data to the API will be done through a single REST call from the application's backend to the /classifications/ endpoint of the API. Being that a Researcher cannot submit a classification without being logged into an authenticated account, the request will have to do no extra work to get authentication information. As a result the submitted request for updated classification will contain: the updated information (in JSON format), the user's identifying information, and a header containing their authentication information.

API Communication with Machine Learning

Since the Moth Classifier API and the machine learning module are separate processes, there needs to be a way for these two to communicate. To solve this problem we will utilize the Publish/Subscribe messaging paradigm with Redis as the message broker as displayed in Illustration 3. The API and machine learning module will communicate through two channels. One is the "job_issuer" channel and the other is the "job_response" channel.

Once the API receives a job and a celery worker is spawned to handle the job, the API will publish a message including the image url and machine learning model to use on the "job_issuer" channel. The machine learning module will be subscribed to this channel and will receive the message. Once the message has been received, the machine learning module will run the image through the specified machine learning model.

Once a prediction has been made, the machine learning module will publish the output on the "job_response" channel. The Moth Classifier API will be subscribed to this channel. When it receives a message from the "job_response" channel, the API will go ahead and update the job status to "finished" and update the relevant classification.

The machine learning program will receive a ZIP file from the API through a URL to train the model. The ZIP file will contain folders of classified moth images. All classified images of the same moth species will be in the same folder. The folder will be named after the species of the images inside of it. The ZIP file will also contain a spreadsheet file, metadata.csv, that will hold a list of all of the folders and the number of pictures that are within each folder. Additionally, it will hold the value of the boolean "is_training". This boolean will be extracted from metadata.csv to determine whether the dataset being sent from the API should be used to train the model or to be classified by it.

The machine learning model will be coded through Python and Tensorflow, an open-source machine learning library. The ZIP file delivered from the API will be interpreted by the machine learning code through the Keras, an interface for the Tensorflow library. Specifically, the function keras.utils.get_file. This function downloads a file from a given URL and saves the downloaded information at a desired file location. The downloaded files will then be interpreted by the function keras.preprocessing.image_dataset_from_directory. This function will organize a dataset from the downloaded images, and it will use the file names as class names for training the model.

Mobile Application Design

Users, both researchers and non-researchers, will have access to the mobile application on either an iOS device running any version of iOS equal to or greater than iOS 9.0 or any Android device which is running SDK version 21 (Android 5.0, Lollipop). Aside from the login page available to all users, the application will have five UI screens available to the class of users designated as researchers to interact with, and four UI screens available to the remainder of the accounts, the non-researchers, to interact with.

For both classes of user, the default screen will be the image submission screen, called the "Submission View" from now on. On the Submission View of the app, there will be a button which will trigger a popup dialog with instructions on what make up an ideal image submission with regards to the Discover Life project. Additionally, there will be a set of buttons which will allow the user to open a dialogue to select an image from their gallery or take a picture with their phone camera. Once images are selected by the user, there is a back end process that does all of the data scraping and collection into an object which will be used later once the user actually submits their image. From the submission screen, there is a menu bar at the bottom of the screen which will display icons to navigate to the other screens within the app. For both researchers and non-researchers there will be icons for About, Notifications, and Profile screens. Researchers will also have an option for the Reclassify screen.

On the Notifications Screen View, called the "Notification View" from now on, users will be able to view the status of the "jobs" relating to that user's submissions of images. On that view they will be able to interact with each specific submission to expose a popup with information regarding the general state of their classification. On that popup, they will be able to note the submission's ID number, its current status, as well as any available information regarding its classifications. No information can be updated or changed directly from the Notifications View and only can be viewed, regardless of account type.

The mobile application will be notified through the use of Firebase Cloud Messenger (FCM) when there are any updates regarding an individual user's classifications. Notifications, as discussed above in the system design section, will be generated by the API upon successful completion of a classification. The API will make a call to a Google Cloud Function and provide information regarding the results of the classification, and at that point the GCF will trigger FCM to send an asynchronous notification to any devices associated with the account which submitted the photo for classification. The notification will contain information about how certain the machine learning model's prediction was, as well as the results (i.e. was the photo determined to be a moth and if so, what species was it labeled as). Alternatively, the notification may contain a special flag which indicates to the app that the image will need to undergo human review.

The next view exposed to all users is the Profile Screen. On the profile screen, users will be able to view and edit their personal information (name, email). Additionally, if the user's current account class is non-researcher, they will have access to a button which will allow them to initiate the process of applying for a researcher account. They will be required to submit the name of the research project they are working on as well as any other information that they can provide to help administrators come to a decision about whether or not that user should be classified as a researcher.

The last view, exposed only to Researchers, is the Reclassification Screen. On this screen, there is a scrollable Grid View of images which are needing to be manually reclassified. There will also be the ability to apply filters in order to only show images after a certain date or in a certain specified location. Each of the tiles in the Grid will be a clickable card which, when tapped, will expand to a dialogue which will display: the image, the current classification data, and a form at the bottom which will be used to update classification. Once a researcher adjusts and submits a reclassification on an image, it will no longer be viewable under the reclassify tab. Additionally, the person who submitted the image will be notified of the update by the FCM as discussed above.

The following list is a summary of each screen within the application, who can access it, and what functionality is provided within each screen.

Submission Screen: accessible by any type of account

- Submit images for classification
- Review guidelines for "good" image submission
- Select images from either camera or photo gallery

Reclassification Screen: accessible only by Researcher accounts

- View all images flagged as needing manual reclassification
- Filter the list of images based on submission date and/or location
- Submit manual classifications of images based on personal review

Notifications Screen: accessible by any type of account

- View current status of any photo classification submissions
- View image submission id and data regarding each submission

Profile Screen: accessible by any type of account

- View and update personal information (name, email)
- If non-researcher account, apply for a researcher account
- If researcher, can update information about their research mission

About Screen: accessible by any type of account

- View information about the Discover Life project at large
- See links to different resources about the mission and implementation of Discover Life project
- In the future this screen may be moved from the bottom bar and into a popup button in the profile screen, depending on design decisions further down the project road.

Division of Labor

The MothClassifier design consists of three distinct components of equal complexity: mobile application/user-interface development, API, and machine learning. Since the development team is made up of three people, each person will be assigned one of the three components. Reese Jones is responsible for developing the front-end mobile application interface through Flutter. He is also responsible for creating a Google Firebase storage platform that will hold information needed by the mobile application. Abid Ahmed is responsible for creating the API of the entire system. He is responsible for linking the machine learning code to the mobile application through the Django REST Framework. Additionally, Abid will be responsible for the communication between our platform and the Discover Life website. Ben Giangrasso is responsible for creating machine learning code with Python and Tensorflow. Ben will also retrain machine learning models to ensure that the models used in the mobile application will give the highest classification grade.

Project Timeline

The design's modules will be completed by the timeline listed below. Each milestone of the timeline will build up to the initial release of the mobile application on various marketplaces as well as its release to Discover Life.

24 November 2020

- Improved Discover Life web scraper
- Implement design mockup pages within mobile application
- Differentiated app state for researchers
- Serve collections of data from API
- Implement API Authentication
- Write API Documentation
- Train a simple model on Discover Life moth data

09 December 2020

- Basic integration with API and machine learning using a pub/sub pattern
- Basic data flow between API and mobile application using HTTP calls
- Moth classification model with nontrivial accuracy
- Finalize skeleton of all blank and semi-developed screens in mobile application

20 January 2021

- All screens have business logic and data to handle some type of interaction
- API can successfully return data to the Firestore DB and the Google Cloud Messaging service
- API can successfully create jobs for the machine learning model and collect its responses
- Expand classification to include at least one species with reasonable accuracy
- Improve accuracy of moth classification model (separate from species classification)

24 February 2021

- Submitted mobile application to marketplaces for approval
- Create and deploy API to production environment
- Deploy machine learning to production environment
- Machine learning model can identify moths as well as two different moth species with good accuracy

07 April 2021

- Successful data flow from image submission and classification through all components
- Production environment is stable
- Project is packaged to hand off to Discover Life
- Mobile application automatically monitors and reports Crashlytics for bug identification
- Mobile application is published to Play Store and iOS App Store