CENTER FOR SPATIAL INFORMATION SCIENCE AND SYSTEMS

Geofairy iOS/Android App and Server

Software Architecture Document

Version 0.8

07/08/2022

Document Number: 20220708

Contract Number: +1 703 993 6124

Table of Contents

1. Introduction

Geofairy is an award-winning App trying to realize one-stop location based service (LBS) for users to retrieve geospatial information (GI) via their mobile devices. It has 8 source datasets and more than 100 data layers covering the whole globe, including most developing countries. The information types include weather, vegetation, elevation, soil moisture, land cover, atmosphere, and precipitation. For developing countries these information are very helpful but lack or hard to retrieve due to rare open earth observation projects and inappropriate data use. Geofairy could provide everyone in developing country with free geospatial information to help them make decisions about agriculture, biodiversity, climate, disaster, ecosystems, health and weather. This User Manual (UM) provides the information necessary for mobile users to effectively use the Geofairy system on the smart phones.

2. Design

An overview of the GeoFairy system design is shown Figure 1. It is a composition of multiple sub-module components and the connections among them, which are introduced in detail below.

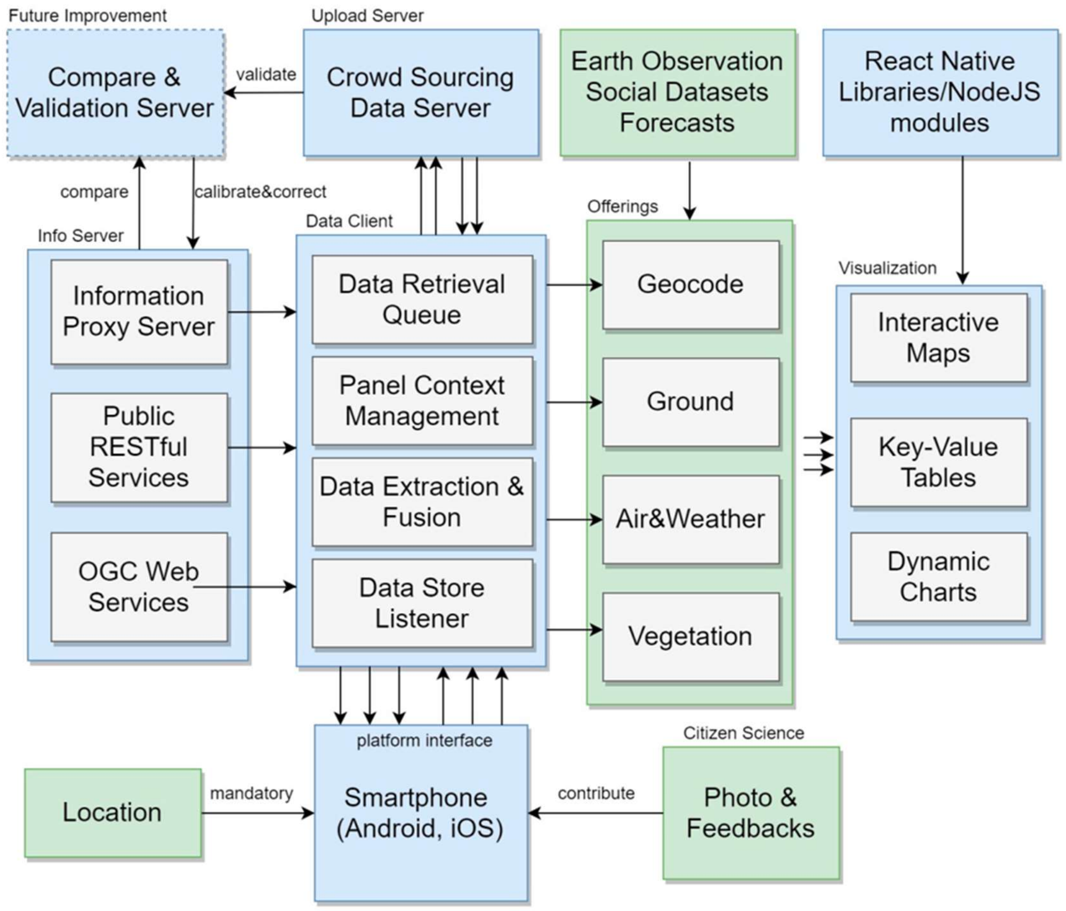

Figure 1. System Architecture (blue boxes - software components; green boxes - data)

2.1 Geospatial Web Service Module

This module refers to the Info Server block in Figure 1 and contains all information services that are used to process data requests from GeoFairy system and send back actual information. To realize the application-level data aggregation, the architecture reuses the existing geospatial web services to fill in this module. There are thousands of web services offering tens of thousands of terabytes of geospatial data that are still growing. However, selecting underlying web services for mobile Apps need extra attention to quality. The web services need to meet several criteria to become qualified as an information source.

2.1.1 High Sustainability

GeoFairy system architecture heavily rely on backend web services. The used web services should have stable long-term availability. The services are better maintained if attended by a specialized person. In that case, only the web services that are backed by well-reputed businesses or reliable government agencies are appropriate to be relied on. Otherwise, the solution code should be prepared for an unexpected situation, such as service offline, interface altering, data gaps, internal errors if those prototype web services from research projects are involved. It is questionable whether the service would persist after the funding is over. Therefore, a modular design is highly recommended to avoid software collapse when exceptions occur during the operation period.

2.1.2 High Throughput & Low Latency

A single server has limited capability in serving user requests. A reliable web service needs solid gateway routers and robust hardware to deal with a large number of concurrent requests. The network interface card (NIC) should have a very high throughput and spare NICs are preferable. If the requests are at an overwhelming level, a load balancer would be recommended and the requests on a single node should not exceed a threshold to provide a low latent response. Memory leaking is a common issue that might cause the entire system to collapse. High memory volume is important to ensure service stability. The volume of Earth observations, especially remote sensing imagery datasets, is tremendous, and processing them need a lot more computational power than normal datasets. Powerful hardware and robust software are the prerequisite to enable low latency and high throughput. These details need to be thoroughly investigated before adopting them.

2.1.3 Interoperable Interface

To address the data heterogeneity challenges, the interface of the candidate web services should be easily interoperable. There should be explicit manuals on calling the services (e.g., Swagger). Standard interfaces such as OGC WMS (Web Map Service), WFS (Web Feature Service), WCS (Web Coverage Service) are preferred. RESTful web services with detailed documentation are also recommended. OpenAPI Specification v3 was released as a milestone for the API developer community, to adopt as a common standard service interface. More standard web services are forthcoming and could be easily integrated via these standards.

2.2 Communication Module

This module refers to all arrows and their associated interfaces in Figure 1. All displayed information in GeoFairy system is real-time collected from the Internet. It has no built-in data upon the installation. A valid Internet connection is required. For remote sites where the signals might be weak, solutions like offline data storage are discussed. The current design is based on the assumption that there are at least occasional Internet connections to the smartphones.

The communication between client and server follows the standard protocols: HTTP (Hypertext Transfer Protocol). Other lower-level network protocols such as TCP (Transmission Control Protocol) or UDP (User Datagram Protocol) can also be directly used. Today most smartphones are equipped with hardware to support multiple protocols like Wi-Fi (IEEE 802.11), 4G LTE (IMT-2000), and Bluetooth, to transmit data wirelessly. To communicate with the higher-level standard service interfaces, such as REST API, OGC web services, the client and server need to use the same set of protocols for communication between them. This requires a detailed specification of parameters, structures, encodings, algorithms, etc. For geospatial information, further specifications and standards on projections, resolution, timestamp, data format, and metadata are mandatory for the other-side program to correctly decode the information.

Fortunately, OGC, ISO, W3C, OpenAPI, and many other standardization organizations already considered these issues and made a whole set of interoperability standards accordingly. Those web service standards fill in the vacuum of regulating geospatial information communication from the bottom to the top. The interoperability standards cover almost every aspect of the workflow to transmit geospatial information among individual remote devices. The conventional OGC web services offer two types of protocols—plain XML and SOAP (Simple Object Access Protocol). Recent development shows that RESTful web services have become popular, and JSON is now a routine format for information exchange. The popularity of RESTful web services is largely pinned on its simplicity and generality and is limited to several unified conclusive verbs: GET, POST, PUT, PATCH, and DELETE, which cover almost all transaction requirements. In this architecture, the communication between client and server is in multiple formats and should remain flexible, according to real-world situations.

2.3 Mobile Gateway Endpoint Module

The internal design of the app includes four major data-centric submodules (the Data Client block in Figure 1). Each submodule deals with one specific kind of requirement.

2.3.1 Data Retrieval Queue Submodule

This architecture allows the client to talk to multiple web services simultaneously. The data retrieval graph becomes complicated accordingly because of the network uncertainty, hardware/software capacity, and variety of transferred data volume. Generally speaking, the data retrieval speed is determined by the status of the loads on the client and the server, and the network, which could be simply represented using the following equation:

$$T_r = a \times \frac{L_c \times L_s \times O_d}{C_c \times C_s \times Q_n} + e$$

where T_r is the time cost of retrieving data from server to client; L_c and L_s are the workload on client and server; O_d is the complexity of the transmitted data, which is a combined score measuring the volume, dimension, structure, format, encoding, etc.; Q_n denotes the quality of the network between the client and the server; C_c and C_s are the system capacity of the client and the server and could be measured by the maximum data the system can process every second. The coefficient a is a constant number transforming the ratio to a time unit, and e is the uncertain error.

According to the equation, when the two endpoints and networks in between are settled, the time cost is mostly decided by the real-time workload and data complexity. To reduce the latency of the system, the average requests on each server node should be balanced and remain under certain thresholds, the transmitted geospatial information should be simple and low dimensional. However, in the proposed architecture, even though the requests are simple, the client would struggle to handle the parallel requests to various web services. To achieve a smooth use of the client, a queuing system should be placed to avoid long-time freezing. The first step is to send all requests out via asynchronous channels. A queued receiver will start to listen to the response. As more traffic is conducted on the HTTP protocol, the requests would use the AJAX technique. However, WebSocket and other mutual communication protocols would be better for those tasks. For every response received, the queue would assign a receiver to redirect the information to the corresponding data processor (discussed in Section 2.3.3). The receiver queue is loosely coupled with the interface module and would not crash or delay the user interface when some service requests are jammed or failed.

Another important design principle of this module is to reduce the complexity of the transmitted data. A high degree of complexity is one of the significant features distinguishing EO datasets from the others. The organization and storage of EO datasets on the server side could add a big-time cost to the query and computing performances. To avoid the intensive processing burden relocating to the client-side, the data to be transmitted need to be in very fine grain. The data tree structure should be no more than three levels and the information should be divided into the finest grain pieces. Information should be fragmented and the dimension should be lowered. For example, some satellite image products have hundreds of bands and the coordinate system is three or more dimensional (time and height). The best scale of the transmitted data in one transaction is one band value at one location (three values—latitude, longitude, band value). It would not only relieve the burden on the network and shorten the waiting time for the users, but would also improve the system robustness by simplifying the data processing workflow on the client-side.

2.3.2 Data Extraction & Fusion Submodule

The heterogeneity of multisource data makes it difficult to directly use the received data, especially when there are more than two sources for the same data category. For example, the Land Cover category has NASA NLCD (National Land Cover Database), NASA MODIS Land Cover products, USDA (United States Department of Agriculture), CDL (Cropland Data Layer), GLC (Global Land Cover), FROM-GLC, etc. The NLCD, CDL, and FROM-GLC are in GeoTiff format, and the MODIS Land Cover products are in HDF (Hierarchical Data Format). A large portion of the costs in reusing existing services is charged by this step. The common processes to unify the datasets include reprojection, resampling, re-gridding, reformatting, mosaic, merging, getting location reports, etc. Most web services do all the preprocessing work. GeoFairy only needs to retrieve the data via standard interfaces like WMS GetFeatureInfo without worrying about the data processing method. After retrieving the location-based information, GeoFairy needs to do data harmonization, by integrating multi-source data into a co-registered dataset, with the same or compatible spatial resolution and projection. As it is a point-based data processing, the amount of information to be processed by GeoFairy2 is tiny and can be rapidly accomplished by mobile devices.

2.3.3 Data Store Listener Submodule

The results from the extraction module is pushed into the data store module. The data store is a dedicated block in the smartphone’s memory. Each information category has a separate data store. The data store and the user interface are synchronized. Any change in the data store is instantly reflected in the interface. For example, the data store has new weather forecasting information, the interface refresh is triggered instantly, and the new information is directly displayed. A listener is responsible for constantly monitoring the data stores and triggering the rendering of the corresponding interface region. The listener maintains a set of preinstalled rendering functions for various categories of information. Besides the data from the retrieval module, the data store also saves the data about user preference, e.g., users usually turn off some data categories that they are not interested in to keep the interface concise. Unlike the other in-memory data stores that are cleaned after the app is closed, the client data store would persist in a file database on the smartphone's external memory. Data persistence is also the responsibility of the data store listener.

2.3.4 Panel Context Management Submodule

As a general-purpose app, it covers multi-thematic information, and the interface design should contain multiple tab panels to keep the information organized. Each tab panel is thematic and domain-specific. All panels must be about the same location and this module is built to ensure that. Panel context is important to allow each tab panel to manage their data, while keeping consistent on the target location. Every time users switch to other locations (via clicking on the map or entering city names), the tab panels can always quickly turn around and refresh their information to reflect the situation of the last selected location. The context management also serves as a coordinator to link the datasets via their properties. Sometimes the observations and data products do not agree with each other. Context management is responsible to detect the inconsistency and warn about it or discard the believed corrupted information based on quality control results.

2.3.5 Visualization

The visualization of location-linked data is implemented in three major forms—map, charts, and tables (as shown in the Visualization block in Figure 1). The details are introduced in. In this new design, the improvements are mostly done on adjusting the composite and transitioning among the three forms. Notebook-style rendering has become popular, along with the wide adoption of Jupyter Notebook, within both science and industry. Users only need to swipe on the smartphone screen with all information just under their fingertips. The tables, maps, and charts are aligned and fit into a one-column document. Reading geospatial information should feel no different from reading a newspaper.

2.4 Citizen Science Module

One of the major goals of GeoFaiy Version 2 is to engage the public to interact with geospatial information, rather than just receiving the information. It means the data consumers could also become data providers. There are many citizen science Apps out there, and most of them follow a similar design. The sensors on the client devices are connected with the software via system driver libraries and platform interfaces. For example, both Android and iOS have API for a high-resolution digital camera, global positioning system (GPS) sensor, accelerometer, gyroscope, magnetometer, ambient light sensor, and microphone. The data collection function is developed and equipped within the client software or web pages (Figure 2). When users browse around the client, they find it smooth to transit from the role of users to providers. One or multiple customized data servers are set up to receive the data collected by citizen scientists. In this architecture, a similar design is added on top of the previous modules. The client prepares a few submission forms and embed navigating buttons into the data forms. In the submission forms, users could enter their observations associated with photos, location, and other sensed data. Two server-side programs are deployed to receive the data submitted from the clients (the Crowdsourcing Data Server in Figure 1). One program is a data server responsible for storing and querying the data. People can select their interested citizen science project and the form is dynamically changed for people to contribute observations.

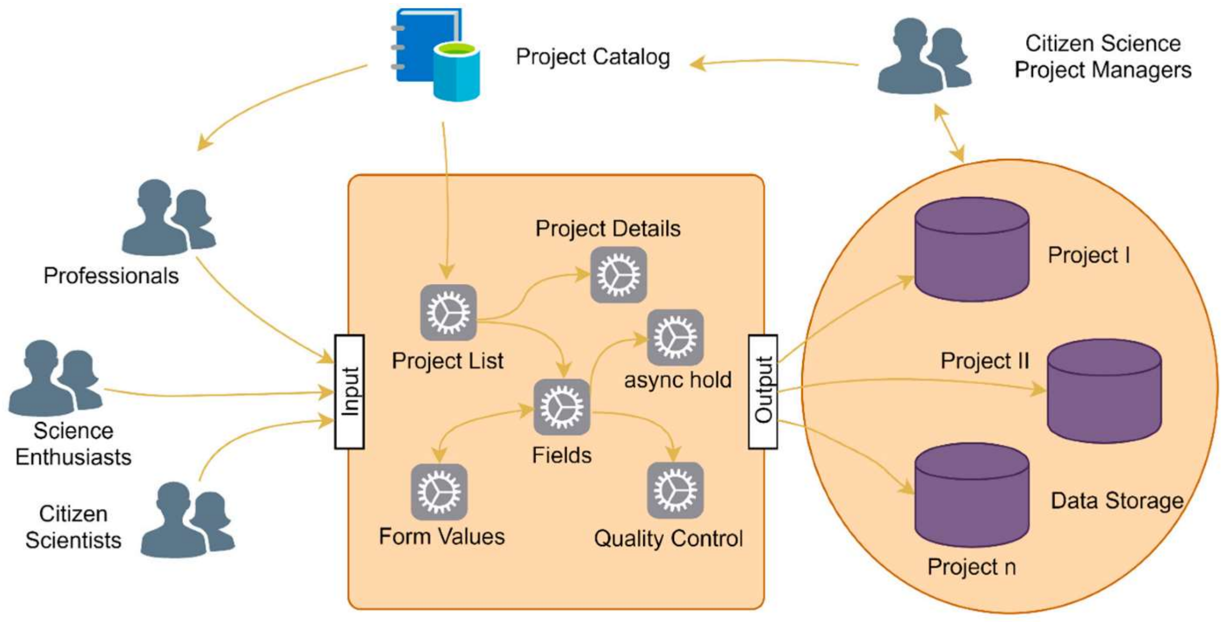

Figure 2. Citizen Science Module Design

Another component is a validation service (the top left blue box in Figure 1) that is responsible for validating the EO products by comparing them with the collected ground truth observations or VGI. If the two disagree, a marker is labeled and the disagreement is recorded for further investigation. If more than three separate individual clients feedback the same disagreements, the validation server places a change request to the EO data server for improvements. The VGI is attached as the new ground truth. If less than three separated feedbacks or the disagreement feedbacks are inconsistent, the request is withheld until enough consistent observations are retrieved. Every client is able to browse its submitted VGI and might also have access to the VGI submitted by other people. All stored VGI is processed to remove personal information about the submitter on behalf of user security. The collected VGI is open and freely accessible via the crowd data server and could benefit tens of thousands of researchers in many disciplines.

The system is composed of client and server module. Server side is maintained by Center for Spatial Information Science and Systems (CSISS) in George Mason University. Users only need download the client App onto their devices. Geofairy server resides in GeoBrain cloud which is a private cloud hosted also in CSISS. Geofairy client is a uniform mobile-style interface, which provide three modes allowing users to view the information from various perspectives. To run Geofairy App, a smart phone with no less than 1GB memory, 1GHz CPU and 1GB memory is required.

The born of Geofairy is inspired by our daily difficulties in gathering and retrieving all kinds of spatial information in practice. Each App is designed to provide some specific information in a relatively static way, e.g., Google Maps for maps and satellite images, AccuWeather for current weather and weather forecast, Climate FieldView for agricultural field related information and ArcGIS for spatial data viewing and analyzing. The Apps have different transfer channels and separated outlets. Users have to download and install an App to acquire the contained information at one time. Besides, extra operations are often needed such as signing up, subscribing services, learning user guide and formalizing recognizable requests. It is complicated and very inconvenient for most users. A simplified App one-stop serving all kinds of GI has been widely recognized as a public desire, especially in emergent scenarios like response actions to disasters like earthquakes, flooding, wildfires, and hurricanes. However, there are very few progresses towards this direction yet. The main challenge comes from the high heterogeneity and poor interoperability of the involved data and service interface.

3. Implementation

The GeoFairy system consists of two types of main components, GeoFairy App and GeoFairy Ground Truth Server. The relationship between GeoFairy App, GeoFairy Server, and Data Sources is shown in Figure 3.

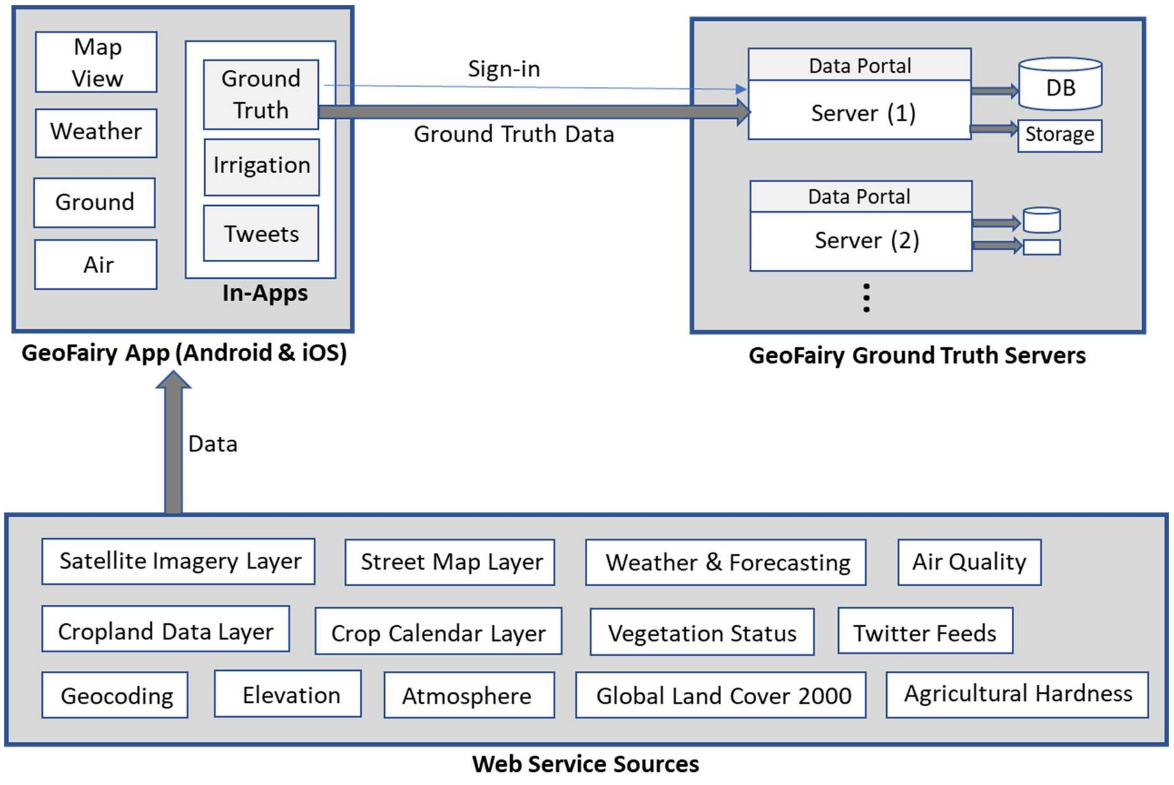

Figure 3. Relationship between GeoFairy App, Ground Truth Servers, and Data sources

3.1 Interrelationship between Subsystems

A GeoFairy App can select one of listed GeoFairy Servers and then submit ground truth data to the server. GeoFairy App can show information such as weather, air quality, ground status, which provided from various data sources by using web service connection.

3.1.1 Acquisition of Authority

During the initializing process of GeoFairy App, the App acquires an access authority from the selected GeoFairy Server with an account. If user didn't fill in any account information, the App will use a default account. The authority needs to communicate between GeoFairy App and Ground Truth Server. The authentication and access control are implemented based on Spring Security.

3.1.2 Web Service Sources

GeoFairy App would directly query third-party web services without going through the GMU server to avoid potential bottlenecks on performance, relieve the burden on the proxy server, accelerate the information loading, and eventually address the data heterogeneity challenges. Table 1 lists most of the datasets displayed in GeoFairy App. As shown in Table 1, the GMU server only provides crop-related information via WMS. Other web services are backed by renowned federal research institutes or commercial companies. In addition to WMS, GeoFairy App also uses REST APIs. The retrieved information is filtered and fused on GeoFairy App by pre-defined rules to only display information with high quality, resolution, accuracy, and value. Both the GMU server and GeoFairy App are operationally maintained, and new versions are released regularly, with additional functionality and bug fixings.

Table 1. Data list and web service sources using in GeoFairy

Dataset Name	Web Service Provider	Interoperability Protocol
Satellite Imagery Layer	Google & Apple	Google Tile/Apple Tile API
Street Map Layer	Google & Apple	Google Tile/Apple Tile API
Weather & Forecasting	NOAA & OpenWeather	NWS API/Weather API
Cropland Data Layer	GMU	WMS
Crop Calendar Layer	GMU	WMS
Twitter Feeds	Twitter	Twitter API
Vegetation Status	GMU & NASA	WMS
Air Quality	World Air Quality Index	WAQI API
Geocoding	Google	Google API
Elevation	USGS	WMS
Atmosphere	NASA	WMS
Global Land Cover 2000	JRC-IES	WMS
Agricultural Hardiness	USDA	GIS REST API

3.2 GeoFairy App

GeoFairy App is developed and built based on *Expo*. *Expo* is a system that allows people to build mobile Apps for iOS and Android using just one JavaScript codebase. Expo is based on React Native by *Meta Platforms Inc*. A top-level software architecture of a GeoFairy App is shown in Figure 4.

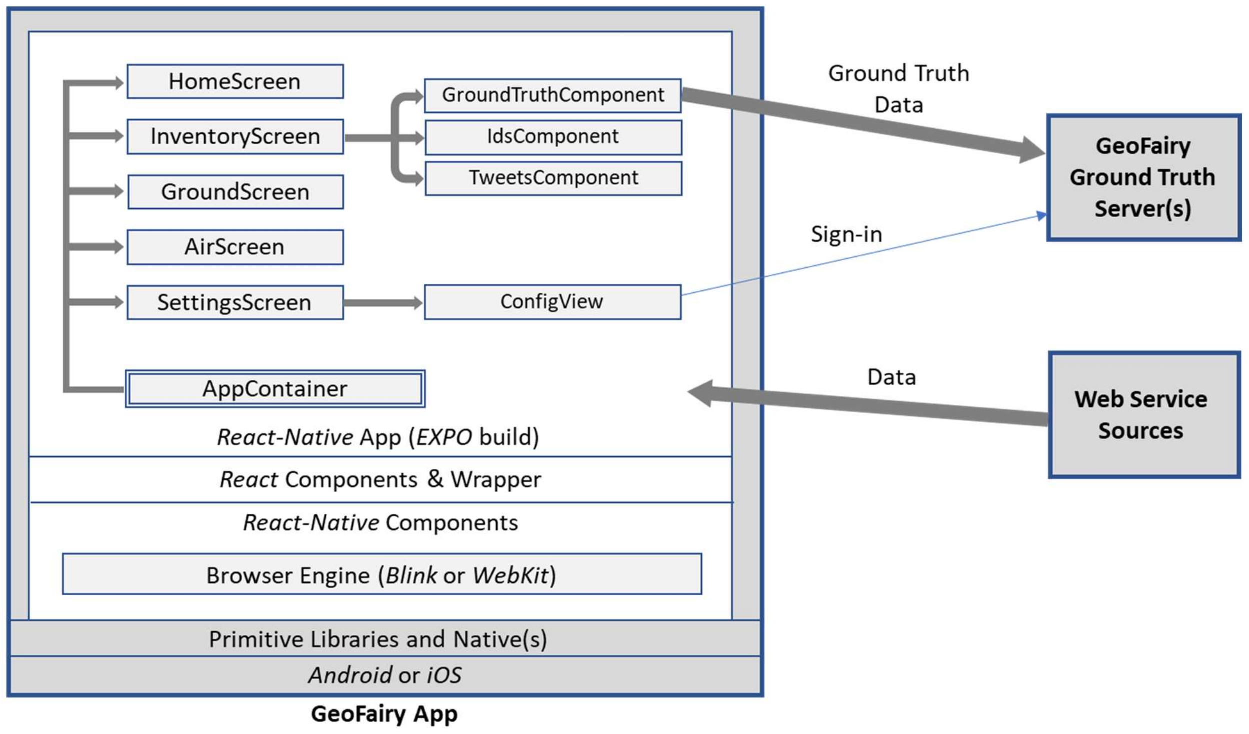

Figure 4. Software Architecture of GeoFairy App

3.2.1 Screen Views

The software architecture of the GeoFairy App is based on React Native. The App view is initialized by React Navigation container which is built by *Expo*, *Software Mansion*, and *Callstack*. The container has 5 inner navigators named *HomeScreen*, *InventoryScreen* (for Utilities), *GroundScreen*, *AirScreen*, and *SettingsScreen*, which are responsible for each 5 main functions of the App.

3.2.2 User Credentials

After loading *SettingsScreen* navigator component, the component is starting to initialize the App configurations. One of important step is acquiring user credential from a GeoFairy Ground Truth Server with the last stored sign-in status. If the last sign-in status was no sign-in status ether signed-out or never sign-in, the App will initialize with sign-in with a default user. The user credential is required to communicate with a GeoFairy Ground Truth Server.

3.2.3 Utilities

The *InventoryScreen* (for Utilities) is designed as an inner-App container, and currently installed 3 inner-apps, named *GroundTruthComponent* for Ground Truth inner-App, *IdsComponent* for Irrigation inner-App, and *TweetsComponent* for Tweets inner-App. Each inner-App icon is wrapped by React Native's TouchableOpacity wrapper. Touch handling of each wrapper switches its inner-App visibility.

Ground Truth inner-App supports submitting ground truth data to a Ground Truth server. User can select one project from listed projects which are allowed to the sign-in user, Input fields in a project are defined by an administrator when the project was created or modified.

3.2.4 Data Sources

Inner communication between a GeoFairy App and all data sources is using Ajax by browser engine. A GeoFairy App source codes is written by JavaScript, and they are transferred as a native binary App working on Android or iOS. The JavaScript codes are running on browser engine which is invoked by *WebView* framework by Android or iOS. Because of security reason, browsers restrict cross-origin HTTP requests, but *WebView* can allow to fetch cross-origin resources by changing its options. That's why GeoFairy can fetch data from various data sources via Ajax connection without the help of any proxy, even though the App codes are written by JavaScript. All data sources are described in Table 1 above.

3.3 GeoFairy Ground Truth Server

GeoFairy Ground Truth Server is a Spring Boot Application using Spring Web and Spring Data JPA database connectivity. The Spring Web is used to implement Web APIs for communicating with GeoFairy App or Project Management Portal, rather than implements as a Spring MVC framework. Software architecture of the GeoFairy Ground Truth Server is shown in **Error! Reference source not found.**.

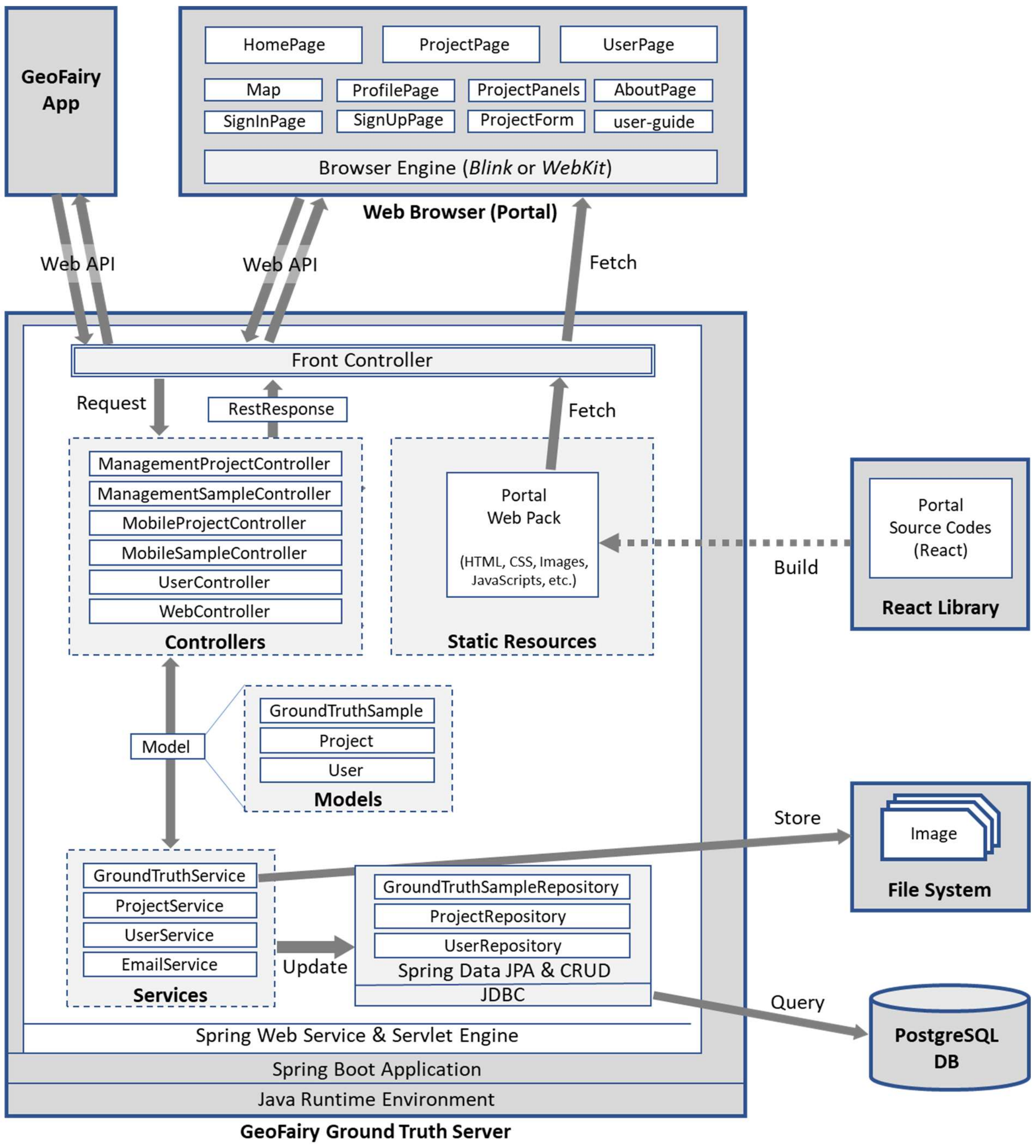

Figure 5. Software Architecture of GeoFairy Ground Truth Server

3.3.1 Web APIs

Spring Web package used on the server is used to implement Web APIs. Each Web APIs is implemented as a Spring Controllers. The controllers consist of 5 controllers for responding portal accesses and 2 controllers for responding mobile App accesses.

3.3.1.1 ManagementProjectController

ManageProjectController is for responding to manage project requests from GeoFairy Project Management Portal. The Web APIs mapped by this controller is shown in Table 2.

Table 2. Web APIs in ManagementProjectController

Web API	Method	Response
/management/projects	GET	Projects
/management/projects	POST	New project ID
/management/projects/{projectId}	GET	A project
/management/projects/{projectId}	PUT	Project ID (OK)
/management/projects/{projectId}	DELETE	Deleted project ID
/management/projects/{projectId}/adminEmails	GET	Admin email
/management/projects/{projectId}/clone	GET	New project ID
/management/projects/{projectId}/images	GET	Image DB records
/management/projects/{projectId}/join	GET	Project ID (sent join requesting)
/management/projects/{projectId}/samples_agg	GET	Agg results set
/management/projects/{projectId}/users	PUT	Project ID (user added)
/management/projects/{projectId}/userEmails	GET	User email

3.3.1.2 ManagementSampleController

ManageSampleController is for responding to manage ground truth sample requests from GeoFairy Project Management Portal. The Web APIs mapped by this controller is shown in Table 3.

Table 3. Web APIs in ManagementSampleController

Web API	Method	Response
/management/groundtruth	GET	Ground truth samples
/management/groundtruth_zip	GET	A zip file (packed Image files)
/management/groundtruth/{project_id}/csv	GET	A CSV file (ground truth samples)
/management/groundtruth/{project_id}/pdf	GET	A PDF file (ground truth samples)
/management/images/{filename}	GET	Image binary
/management/sample/{id}	DELETE	Delete Status
/management/sample/{id}	GET	A ground truth sample
/management/sample/{id}	PATCH	Update Status

Web API	Method	Response
/management/sample/{id}/nearby_image	GET	Image file information list
/management/sample/{id}/sample_edit_images	GET	Image file information list

3.3.1.3 UserController

UserController is for responding to manage user account information from GeoFairy Project Management Portal. The Web APIs mapped by this controller is shown in Table 4.

Table 4. Web APIs in UserController

Web API	Method	Response
/loginContext	GET	Signed-in user information
/register	POST	Created user ID
/users (with ID and/or Email)	GET	A found user information
/users (no parameter)	GET	All user's information
/users	POST	Created user ID (same as /register)
/users/{email}	DELETE	Delete user status
/users/{id}	PUT	Replace user status
/users/{id}	PATCH	Update status (Active and Role)
/users/{id}/password	POST	Status (user's password matching)
/users/message	GET	All messages of signed-in user
/users/projects/approve	POST	Status (added user to join a project)
/users/projects/reject	POST	Status (rejected user of joining request)

3.3.1.4 ForgotPasswordController

ForgetPasswrdController is for responding to manage user's password on GeoFairy Management Portal. The Web APIs mapped by this controller is shown in Table 5.

Table 5. Web APIs in ForgetPasswordController

Web API	Method	Response
/forgotten_password/{email}	GET	Status (sending reset email)
/reset_password	POST	Status (changing password)

3.3.1.5 WebController

WebController is for responding to redirect from given URL paths on GeoFairy Management Portal. The Web APIs mapped by this controller is shown in Table 6.

Table 6. Web APIs in WebController

Web API	Method	Response
/portal	GET	Forward to "/portal/index.html"
/error	GET	Forward to "/portal/index.html"
/portal/reset_password	GET	Forward to "/portal/index.html"

3.3.1.6 MobileProjectController

MobileProjectController is for responding to manage requests projects information from GeoFairy App. The Web APIs mapped by this controller is shown in Table 7.

Table 7. Web APIs in MobileProjectController

Web API	Method	Response
/projects	GET	All public projects information
/projects/{projectId}	GET	A project information
/projects/{projectId}	DELETE	Delete status
/projects_auth	GET	All accessible projects of signed-in user

3.3.1.7 MobileSampleController

MobileSampleController is for responding to manage requests about samples of project from GeoFairy App. The Web APIs mapped by this controller is shown in Table 8.

Table 8. Web APIs in MobileSampleController

Web API	Method	Response
/groundtruth	GET	All ground truth samples by project ID
/groundtruth	POST	Saved a ground truth sample ID
/groundtruth_sec/{token}	POST	Saved a ground truth sample ID
/images	POST	Submit image Status
/images/{filename}	GET	Image binary

3.3.1.8 Security Configuration

Most of all Web APIs requires user credentials. It is defined by a HTTP security setting of Spring Web infrastructure, and there is no user defined controller for this action. The sign-in is done by requesting the following Web API described in Table 9.

Table 9. Web API for Security Configuration

Web API	Method	Response
/login	All	Authentication result

3.3.2 Database Connectivity

Data manipulation of the server is implemented by using Spring Data JPA, which makes it easy to easily implement Java Persistence API (JPA) based repository. All database queries, including create tables, are managed by Spring Data JPA layer. The GeoFairy server is just using Spring JPA Repository APIs and does not access the tables directly by QUERY strings. All used database schemas in the GeoFairy server are shown as an ER diagram in Figure 6.

There are two main tables in the schemas: *project* and *user*. The *project* table keeps all projects information both public type and private types. The *user* table keeps all users account information. An ownership mapping between project and user is described in table *project_admin*. Each record has two columns, project ID and user ID which are a primary key of the two tables. Similarly, a joining relationship between project and user is described in table *project_user*.

Ground truth samples information submitted by GeoFairy App are keeping in the table *ground_truth_sample*. Each record in the table can have an image path which is the location of stored image file. Stored image is kept in a directory in the server storage by the server configuration (*geofairy.data.path*=…). The default location is "*/Data*". The table *user_messages* is for keeping user's messages such as request a join to a project by each user.

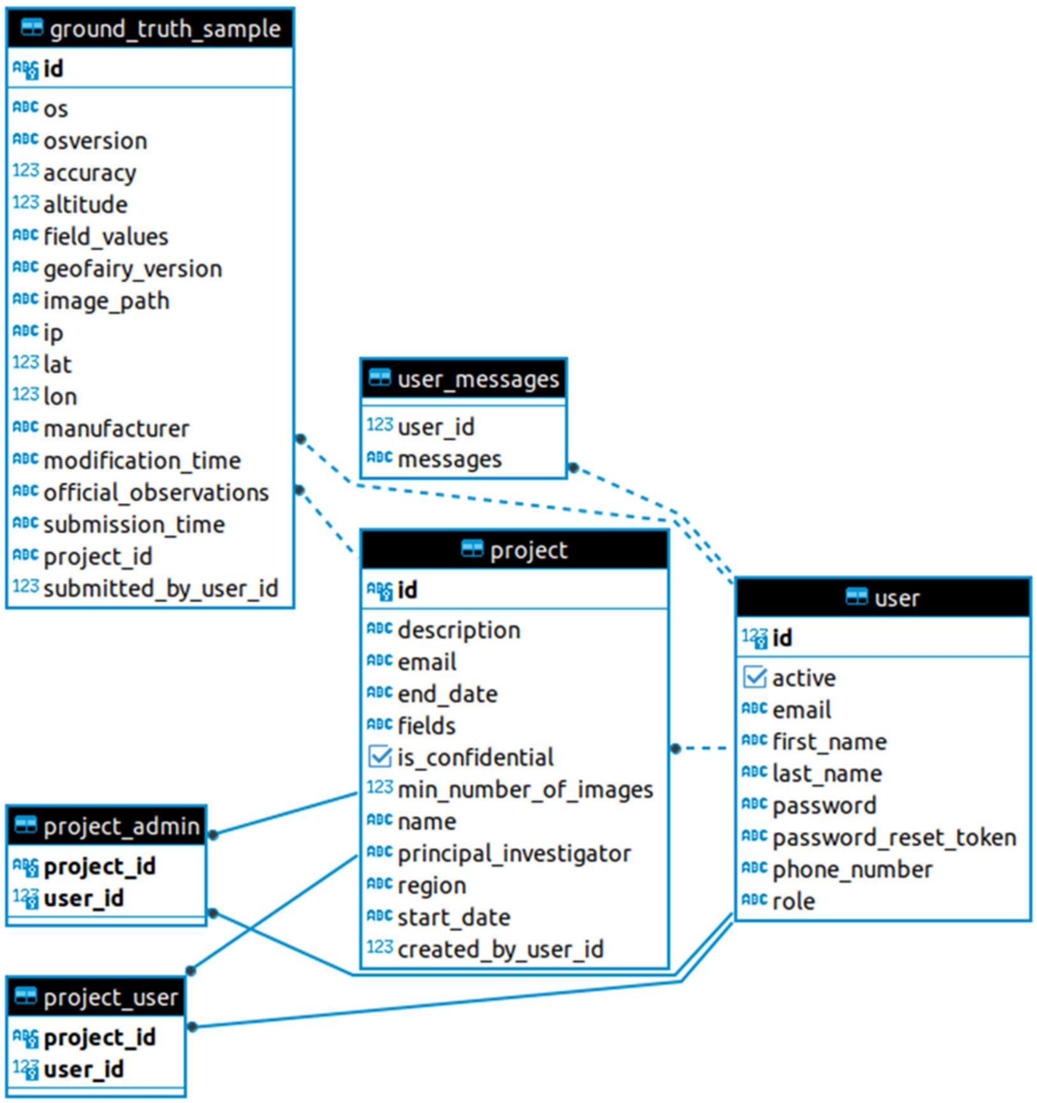

Figure 6. ER Diagram for GeoFairy Ground Trust Server

.

3.3.3 GeoFairy Project Management Portal

The GeoFairy server has an its own portal for managing projects and registered users. The portal is implemented an independent React web pages, instead of using Thymeleaf templates. The portal communicates with its server by Web APIs during runtime, not by using dynamically generated web pages. All the Web APIs is listed in section 3.3.1.6 and 3.3.1.7. An overview of the software architecture of the GeoFairy Project Management Portal is shown in Figure 7.

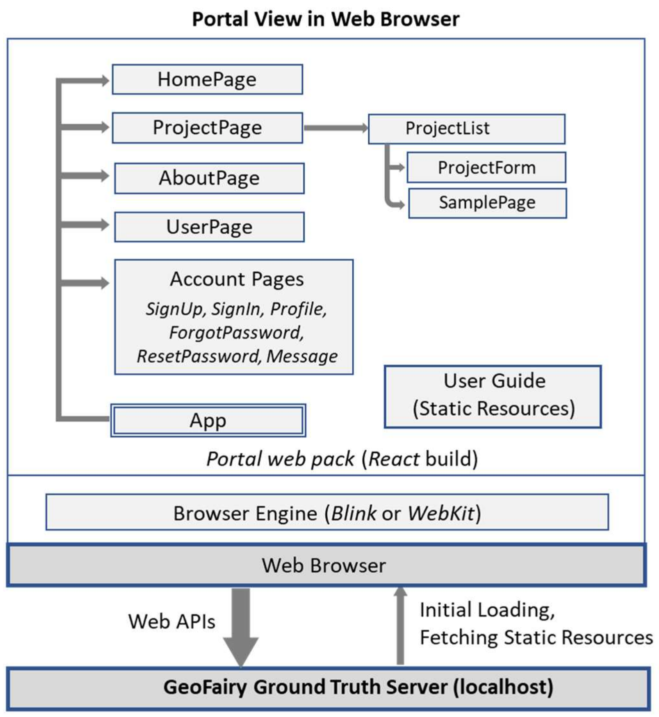
Portal View in Web Browser
HomePage
ProjectPage
ProjectList
ProjectForm
SamplePage
AboutPage
UserPage
Account Pages
SignUp, SignIn, Profile, ForgotPassword, ResetPassword, Message
User Guide (Static Resources)
App
Portal web pack (React build)
Browser Engine (Blink or WebKit)
Web Browser
Web APIs
Initial Loading, Fetching Static Resources
GeoFairy Ground Truth Server (localhost)

Figure 7. Software Architecture of GeoFairy Project Management Portal

There are three main pages are in the portal: *HomePage*, *ProjectPage*, and *UserPage*. The *HomePage* is a starting point page. The *ProjectPage* is for project management, including project creating, updating, deleting. The page also supports requesting to join private projects. The UserPage is for managing user accounts such as changing role, active status, and account deletion.

The portal is written as a React web pages, and the portal codes are web-packed by React tools. The packed codes are put into the server's static resource folder, named "portal". After that, GeoFairy server will be packaging as a Spring Boot application.

4. Troubleshooting & Support

4.1 Error Messages

If GeoFairy App complains about location error, make sure you have enabled location services of your phone.

If GeoFairy App shows blank, double check your Internet connection.

Still got error messages, please contact us.

4.2 Special Considerations

Users with disabilities may contact us to tell your difficulties in using GeoFairy App.

4.3 Support

Table 10 – Support Points of Contact

Contact	Organization	Phone	Email	Role	Responsibility
Liping Di	*CSISS, GMU*	*+1 703 993 6114*	*ldi@gmu.edu*	*Director*	
Ziheng Sun	*CSISS, GMU*	*+1 703 993 6124*	*zsun@gmu.edu*	*Development Leader*	

Appendix A: Record of Revision

Table 11 – Record of Changes

Version Number	Date	Author/Owner	Description of Change
0.8	*07/08/2022*	*Gil Heo (CSISS)* *Ziheng Sun (CSISS)* *Liping Di (CSISS)*	*Initiated version*