Contents

Overview

The Plai Platform API is the protocol used by client apps to read or save data on the server. This document is a developer's reference for the network endpoints. It is intended for client app developers (iPhone, Android, and web interfaces).

REST Introduction

The Plai Platform API is a REST JSON API. This means it uses the REST protocol (which is simply the HTTP protocol), and that the data is formatted as JSON arrays. It is used by the client applications to get, save, or update database records. It is also used to upload files.

REST is a request-response protocol that is built on top of HTTP. It makes use of HTTP's verbs, urls, and upload formats. The responses from the server are all JSON-formatted arrays of key/value pairs, or else ordered lists (arrays) of values without any key. The arrays can be nested, creating trees of arrays.

iPhone Swift

iPhone Swift can handle the HTTP (verbs, urls, and upload formats) natively. Here are some examples, but you would still need to parse the JSON responses somehow:

- https://stackoverflow.com/questions/24321165/make-rest-api-call-in-swift

iPhone Swift REST code can be made simpler by using the first two libraries mentioned in the following article. The first library handles the HTTP more simply than native code, and the second one handles reading the JSON-formatted responses and accessing the variables.

- https://infinum.co/the-capsized-eight/top-10-ios-swift-libraries-every-ios-developer-should-know-about

There are other libraries available for Swift also:

- https://github.com/bustoutsolutions/siesta

Android Java

Android supports both HTTP and JSON natively. Here is a tutorial:

- https://code.tutsplus.com/tutorials/android-from-scratch-using-rest-apis–cms-27117

However Android can be made a little simpler by the use the android-async-http library:

- https://stackoverflow.com/questions/29339565/calling-rest-api-from-an-android-app

Another library is called Retrofit. It maps the network endpoints to a Java "Interface". This may or may not be helpful.

- https://android.jlelse.eu/consuming-rest-api-using-retrofit-library-in-android-ed47aef01ecb
- https://inducesmile.com/android/android-retrofit-2-with-json-api-example/

HTML Vue with Vuex

Vue with Vuex can use the christianmalek/vuex-rest-api library (or just raw Axios):

- https://github.com/christianmalek/vuex-rest-api

REST HTTP Verbs

In REST, you must first specify which "verb" you wish to use. These verbs are part of the HTTP specification, and with REST they are supposed to be meaningful. The REST verbs are GET, POST, PUT, PATCH, and DELETE.

- **GET**: Use to read data (either a searchable list, or a single database entry)
- **POST**: Use to create new data (incl. file uploads)
- **PUT**: Used to update or edit existing data
- **PATCH**: (this is the same as PUT, just use PUT)
- **DELETE**: Delete a database entry

With the network endpoints listed below, be sure to use the correct verb as specified. Using the wrong verb will generally result in an error, or, if you accidentally specify DELETE instead of PUT, it could delete the data you were trying to update. Your HTTP/JSON library will provide an interface to specify which verb to use for each request.

REST HTTP URL

The URL is how you can specify the network endpoint you with to use. For example:

```
https://40.78.81.126/api/events
```

Sometimes the network endpoint will have parameters. For example, a POST to this endpoint allows a user to join an organization (assuming it has open enrollment):

```
https://server/api/organizations/{organization}/join/{user}
```

The "organization" and "user" parameters (shown above with curly braces) must be supplied by the client app. In all cases, the parameter will be a positive integer, which is the unique id. For an admin of organization with id=7 to add a user with id=101, the URL would be:

```
https://40.78.81.126/api/organizations/7/join/101
```

The server part will be specific to which server you are talking to. It may be a development machine, like this:

```
https://40.78.81.126
```

or it may be the official Plai Platform Server:

```
https://plai.today
```

In the documentation below, the server portion is left out of the documentation. Only the path portion of the URL is shown, like this:

```
/api/organizations/{organization}/join/{user}
```

But any client code will need to specify the full server IP address or server name to the HTTP/JSON library.

REST Optional Arguments

Regardless of which HTTP verb is used (GET, POST, DELETE, etc.), various data or arguments can be uploaded as part of the request. Sometimes this is used to upload new data, like new Event details, but sometimes it is used to specify optional search filters, for example:

```
GET https://40.78.81.126/api/events?q=tennis
```

Here, the search string 'q' is passed formatted as an HTTP "query string" variable. If this request was using POST, it would be in the HTTP post-data format instead. Your HTTP/JSON library will provide an interface to specify your arguments as key-value pairs, and it should handle the formatting for you automatically. The server can handle multiple input formats, including the GET query string, POST "form-data", and PUT "x-www-form-urlencoded". (Since the PUT verb only supports "x-www-form-urlencoded", it is recommended to use "x-www-form-urlencoded" by default.)

REST Headers

Part of the HTTP (and REST) protocol is the list of headers. There are always two headers required for the network endpoints:

```
Accept: application/json
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwczpcL1wvMTAuMC4zI
```

The Authorization header will have a different "Bearer token" for each login; the above random string is just an example. But the other two headers should appear exactly as shown: "Accept: application/json".

There is also a third header required if using PUT requests:

```
Content-Type: application/x-www-form-urlencoded
```

This third header is necessary because PHP does not support the default POST format of “form-data” for the verb PUT. The header above tells the server that the new data you trying to PUT on the server is formatted in “x-www-form-urlencoded” instead of “form-data”.

For more inforation, see: https://laravel.io/forum/02-13-2014-i-can-not-get-inputs-from-a-putpatch-request .

Alternatively, if it is easier to use “form-data”, you can use a work-around where you skip the third header, and use the verb POST instead of PUT, but also set a magic variable named “_method" with the value of “PUT”. For more info, see:

https://laravel.com/docs/5.7/routing#form-method-spoofing

NOTE: This “method spoofing” trick may be necessary if uploading a file in a PUT request.

Use whatever is easiest. The Postman developer utility has a checkbox for “x-www-form-urlencoded”, and it automatically adds the corresponding header.

Your HTTP/JSON library will provide an interface to specify the headers. Most libraries have a way to specify the Authorization header to be automatically included for every outgoing request, since that is such a common need. The “Accept: application/json” may already be a default header, but check your library documentation to be sure.

REST Upload Key/Value Pairs

The REST protocol allows the user to upload key/value pairs of data. In the models shown below, the POST or PUT methods will create or update any named field that is not a computed field (such as “distance” or any included summary counts).

Some URLs accept a file upload. Where specified, those will also contain a “key” name.

Using Postman

There is a free developer tool called **Postman**. It can be used to hand-craft REST/HTTP requests, and then examine the responses from the server.

Postman is a handy tool to bypass any code you have written, to make sure your HTTP request works correctly on the server. It also allows you to examine the responses to see what the data looks like. Postman runs on Linux, Mac, and Windows.

https://www.getpostman.com/

Postman allows you to specify the HTTP verb, the URL, and any key/value data pairs, as well as headers. If you click on the "bulk edit" button, you can copy and paste values into a text edit field, in the format of one "key: value" per line.

Before writing client application code, use Postman to experiment with the network endpoints and their arguments.

Auth with Postman

When using Postman to test or debug, you must manually paste in the headers:

```
Accept:application/json
Content-Type:application/x-www-form-urlencoded
Authorization:Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwczpcL1wvMTAuMC4zL
```

The first two lines are always the same. But the third line contains an auth "bearer" token. It is the security key you get after logging in.

This must be a valid token, or else you'll just get an error. To get a token on a development server, first create a new user:

```
POST https://40.78.81.126/api/auth/register?email=yourname@gmail.com&password=SuperSecret&na
```

This will create the user with email `yourname@gmail.com`. Use a unique value for the email for your test users, because you cannot use an email address more than once (without wiping the users table clean).

The server will return a Javascript Web Token (JWT) in a JSON response like this:

```
{
    "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwczpcL1wvMTAuMC4zL
    "token_type": "bearer",
    "expires_in": 3600
}
```

The random characters with key "access_token" must then be copied and pasted into your HTTP headers, like this:

```
Authorization:Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwczpcL1wvMTAuMC4zL
```

Every time you log out, or your token expires, you will need to repeat the copy-paste operation from the valid JWT "access_token", to the HTTP `Authorization` header.

This manual copy-paste in Postman is an analogy for what the client applications must do. They must periodically update their token, and add to the `Authorization`.

Since the server uses the JWT token format, which is a well-known standard, the REST / JSON library on your platform may have useful functions for handling the auth token automatically. If not, you'll need to manually copy the token from the JWT response to the headers of your API requests. **The header name is `Authorization`, and the value is `Bearer` (including the trailing single space) followed by the random string of token characters.**

Note that the app will need to handle expired tokens gracefully, and (by default) attempt another login if you get the error status 401 "Unauthenticated." message, like this:

```
Status: 401 Unauthorized

{
    "message": "Unauthenticated."
}
```

HTTPS error handling with Postman

If the server does not have a signed SSL certificate installed, then HTTPS negotiation will fail, and Postman will simply give an error. This is likely to happen in early development. To work around this error, configure Postman by going to **Settings > SSL certificate verification: OFF**.

Your client REST/JSON library will need a similar option to ignore SSL errors for early development. Once a signed SSL certificate is installed, and the server changes from https://40.78.81.126/ to https://plai.today/ (with a signed SSL certificate), then the SSL error checking should be re-enabled in any client apps, for stronger security.

Models

Overview

The server manages multiple tables, and relationships of tables. The network endpoints provide access to this data, using the REST protocol. The data is

returned as arrays of key/value pairs, which can be considered the data models of the Plai application.

The primary models are **Events**, **Users**, and **Organizations**. These can be related to one another in various ways; for example, a user can be the creator/owner of an event, or an attendee of an event, or a member of an organization. An organization can post events that are visible only to only their members, or to all users.

Users

A User refers to an authenticated user that can log in to the app. A user can create events, organizations, and so on.


```
"data": {
    "id": 49,
    "name": "Beaulah O'Keefe II",
    "nickname": "ashly00",
    "avatar_url": "public/avatars/3b58deb4-8457-4ef6-8eeb-f2acef5ff08f.png",
    "created_at": "2018-09-13 01:45:04",
    "post_status_id": 3,
    "post_status_name": "public",
    "post_status_display_name": "Everyone (General Public)",
    "organization_admin_count": 0,
    "organization_poster_count": 0,
    "organization_member_count": 0,
    "event_count": 0,
    "user_profile": {
        "id": 49,
        "user_id": 49,
        "description": "Hatter: and in despair she put her hand on the back. However, it was
        "image_url": "http://www.ward.info/voluptas-velit-neque-consequatur-delectus-sed-vol
        "video_url": null,
        "uses_calendar": 1,
        "twitter": "http://www.lowe.net/",
        "instagram": "http://www.maggio.org/ad-ex-quis-sed-odit-ut-maxime",
        "facebook": "http://www.reinger.org/eius-praesentium-aperiam-velit-velit-exercitatio
        "gender_id": 3,
        "skill_level_id": 3,
        "age_group_id": 1,
        "location_id": 49,
        "created_at": "2018-09-13 01:45:05",
        "cross_street": "Jordane Expressway & Osinski Orchard",
        "address": "77920 Witting Knoll",
        "address2": "6",
```

```
        "city": "Edwardborough",
        "state": "0",
        "country": "Guinea",
        "postal_code": "31275",
        "cc": "FJ",
        "latitude": "-65.335228",
        "longitude": "137.523728",
        "gender_name": "coed",
        "gender_display_name": "Co-Ed",
        "skill_level_name": "advanced",
        "skill_level_display_name": "Advanced",
        "age_group_name": "any",
        "age_group_display_name": "Any",
        "distance_miles": 7461.194761304259
    },
    "organizations_where_admin": [],
    "organizations_where_poster": [],
    "organizations_where_member": []
}
```


The networks endpoints restrict what a given user may do. A user can't delete another user's event, for example.

User Logins

There are a series of network endpoints dedicated to users registering, logging in, or logging out. These all start with the /api/auth/ prefix, such as /api/auth/register and /api/auth/login.

Once a user registers or logs in, they will be presented with an access token. This token is in the Javascript Web Token (JWT) format, which is a public standard. This is an an example:


```
{
    "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwczpcL1wvMTAuMC4zL
    "token_type": "bearer",
    "expires_in": 3600
}
```


This token must be put into the "Authorization" header for all subsequent requests – this is what allows access to the API. It should have the prefix "Bearer" and then the random string, without any quotes. The token expires every hour and a new token can be obtained at any time using the /api/auth/refresh endpoint.

If you do not have a valid bearer token, you will get a simple JSON response that says this:

```
{
    "message": "Unauthenticated."
}
```

Google Login

Coming Soon! 2018-09-12

NOTE 2018-09-12: The Facebook and Google login URLs will be added soon, in the next revision of this document. They will return a JWT token just as above, but will require the Oauth2 token instead of the username and password.

NOTE: Facebook or Google accounts will supply the Oauth2 token instead of a password, which will be used to get the email address from Google or Facebook.

Facebook Login

Coming Soon! 2018-09-12

Guest Login

Coming Soon! 2018-09-12

User Registration

For a user to register locally with Plai (without using their Google or Facebook login) they must supply a name, email, password, and (optionally) an avatar_url. Existing Plai users must supply the email and password to login.

For local Plai users, a PNG avatar image will be generated for the user if they do not supply an avatar_url – it will be a randomly-colored circle with the user's first two initials in it.

The user's unique login key is their email address. Users cannot register with an email address that has already been used. A successful registration will return a Javascript Web Token (JWT), like this:

```
{
    "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwczpcL1wvMTAuMC4zL
    "token_type": "bearer",
    "expires_in": 3600
}
```

The REST status code will also indicate SUCCESS 200. Clients can use their REST libraries to check for a 200 result code.

An unsuccessful registration will appear with an error message:

```
{
    "message": "The given data was invalid.",
    "errors": {
        "email": [
            "The email has already been taken."
        ]
    }
}
```

The REST status code will also indicate an error code (UNPROCESSABLE ENTITY 422 in the example above).

User Friendship

User accounts can have a "friend" relationship. Users can send a friend request, which can then be accepted or rejected. Users can also unfriend, block, or unblock other users.

If a user sets the privacy setting to "Friends Only" (for an Event or User Profile information), only their friends will be able to see it. Being friends with a another user also allows you to see a list of mutual friends, and friends of friends, which can be used to suggest new friendships to grow the social network. User profiles can also be searched within a given radius, so the client app can suggest new nearby friends with similar activity interests.

There are several network endpoints dedicated to managing friend relationships. These all start with /api/users/friends/.

User Profiles

Users have associated profiles, which store additional user information (such as address, activity types, etc.). Both profiles and events can be flagged as 'Just Me (Draft Edit)', 'Friends Only', or 'Everyone (General Public)'.

A user can always see their own profile. They will be able to see other users' profiles if they are public, or if they are friends and it's posted as 'Friends Only'.

Several of the network endpoints include user profiles in the results. For those endpoints, there will be a key named "user_profile". It will either be null, meaning it is private or you are not a friend, or else it will be a nested JSON array of key/value pairs for that profile.

Here is an example of a user with a hidden profile (user_profile: null, because we are not Dr. Mustafa's friend) followed by a user with a public profile (user_profile: sub-array of key/value pairs).


```
{
    "id": 42,
    "name": "Dr. Mustafa Cummerata IV",
    "nickname": "ebert.darron",
    "avatar_url": "public/avatars/714c1d99-0bdd-417a-9018-0e22f137572f.png",
    "created_at": "2018-09-13 01:45:04",
    "post_status_id": 2,
    "post_status_name": "friends",
    "post_status_display_name": "Friends Only",
    "organization_admin_count": 0,
    "organization_poster_count": 0,
    "organization_member_count": 0,
    "event_count": 0,
    "user_profile": null
},
{
    "id": 34,
    "name": "Dr. Cruz Wilkinson",
    "nickname": "colin28",
    "avatar_url": "public/avatars/370cafd4-dde3-4d45-becc-ee719f3164b5.png",
    "created_at": "2018-09-13 01:45:04",
    "post_status_id": 3,
    "post_status_name": "public",
    "post_status_display_name": "Everyone (General Public)",
    "organization_admin_count": 0,
    "organization_poster_count": 0,
    "organization_member_count": 0,
    "event_count": 0,
    "user_profile": {
        "id": 34,
        "user_id": 34,
        "description": "I don't understand. Where did they live at the place of the trees be
        "image_url": "http://www.roberts.biz/",
        "video_url": null,
        "uses_calendar": 0,
        "twitter": "http://www.lang.com/neque-inventore-repellat-et",
        "instagram": "http://www.glover.com/rerum-ducimus-quas-provident-provident-eveniet-p
        "facebook": "http://stroman.com/quia-est-quidem-ut-quisquam-excepturi-rerum-sunt",
        "gender_id": 3,
        "skill_level_id": 1,
        "age_group_id": 1,
        "location_id": 34,
```

```
        "created_at": "2018-09-13 01:45:05",
        "cross_street": "Morar Cape & Gislason Pines",
        "address": "52478 Hettinger Points",
        "address2": "3",
        "city": "East Berniece",
        "state": "0",
        "country": "Azerbaijan",
        "postal_code": "78248",
        "cc": "GF",
        "latitude": "14.020966",
        "longitude": "-12.324070",
        "gender_name": "coed",
        "gender_display_name": "Co-Ed",
        "skill_level_name": "casual",
        "skill_level_display_name": "Casual",
        "age_group_name": "any",
        "age_group_display_name": "Any",
        "distance_miles": 1284.8547500856578
    }
},
```


Note that the user's name, nickname, and avatar_url is not part of the profile; even users with a private profiles still have a public avatar that can be displayed when listing users in a client application.

Latitude and Longitude

The event, user_profile, and organization structures includes two fields, latitude and longitude. These are used when calculating distances. The events, users, and organizations can be filtered to within a certain radius of a provided latitude and longitude. (Distance is measured in miles, and is calculated by the server using the Haversine formula.)

The user_profile's latitude and longitude is meant as a default, or "home" location, of the currently logged in user. But the API allows any location (such as the user's current GPS location) to be used for distance filters or distance calculations.

To calculate or filter distances according to an arbitrary value, pass the arguments **near_lat** (for latitude) and **near_lon** (for longitude). The server will dynamically calculate the distance to the event, user, or organization. It can also filter those distances to a certain radius, using the **max_miles** argument (more on that below). If no arguments are provided, the latitude and longitude of the user's profile will be used as a default value for the distance calculations.

Events

Events can be posted with a permission of 'Just Me (Draft Edit)', 'Friends Only', or 'Everyone (General Public)'. This is set using the **post__status__id**. (The list of possible values for **post__status__id** can be obtained from the /api/context endpoint.)

Here is an example event detail view, including the list of users going:


```
{
    "data": {
        "id": 50,
        "title": "P90X3 Week 4",
        "description": "Her listeners were perfectly quiet till she shook the house, and wor
        "phone": "686.337.0045",
        "event_email": "reyes62@example.net",
        "url": "http://www.ankunding.org/voluptate-aut-exercitationem-iure-deserunt-est-perf
        "twitter": "http://www.jacobi.com/",
        "instagram": "http://www.anderson.com/",
        "facebook": "http://crona.com/fuga-et-eos-ut-maxime-molestiae.html",
        "image_url": "http://www.littel.biz/magnam-est-velit-dolor",
        "video_url": "https://beier.info/beatae-quibusdam-velit-et-ipsam-qui-accusamus.html'
        "desired_user_going_count": 12,
        "start_datetime": "2017-09-14 01:14:21",
        "end_datetime": "2017-09-14 01:29:21",
        "post_status_id": 3,
        "organization_id": null,
        "organization_post_status_id": 4,
        "user_id": 14,
        "user_name": "Dr. Yasmin Ward II",
        "gender_id": 1,
        "age_group_id": 4,
        "activity_type_id": 4,
        "skill_level_id": 2,
        "created_at": "2018-09-13 01:45:07",
        "updated_at": "2018-09-13 01:45:07",
        "cross_street": "Pansy Views & Streich Lights",
        "address": "7924 Jolie Path Suite 831",
        "address2": "6",
        "city": "Benfurt",
        "state": "1",
        "country": "Maldives",
        "postal_code": "75885-2552",
        "cc": "IQ",
        "latitude": "12.866464",
        "longitude": "-116.166559",
```

```
"user_going_count": 1,
"user_alerting_count": 0,
"gender_name": "m",
"gender_display_name": "Male",
"age_group_name": "youth",
"age_group_display_name": "Youth",
"activity_type_name": "camping",
"activity_type_display_name": "Camping",
"skill_level_name": "intermediate",
"skill_level_display_name": "Intermediate",
"post_status_name": "public",
"post_status_display_name": "Everyone (General Public)",
"organization_post_status_name": "members",
"organization_post_status_display_name": "Members Only",
"distance_miles": 7982.164310259635,
"users_going": [
    {
        "id": 51,
        "name": "Derek Simkowiak",
        "nickname": null,
        "avatar_url": "public/avatars/3f7770c2-8e7f-411f-9639-9ccf4f8a8c99.png",
        "created_at": "2018-09-13 01:45:15",
        "post_status_id": 1,
        "post_status_name": "unposted",
        "post_status_display_name": "Just Me (Draft Edit)",
        "organization_admin_count": 0,
        "organization_poster_count": 0,
        "organization_member_count": 0,
        "event_count": 0,
        "pivot": {
            "event_id": 50,
            "user_id": 51
        },
        "user_profile": {
            "id": 51,
            "user_id": 51,
            "description": null,
            "image_url": null,
            "video_url": null,
            "uses_calendar": null,
            "twitter": null,
            "instagram": null,
            "facebook": null,
            "gender_id": null,
            "skill_level_id": null,
            "age_group_id": null,
```

```
                    "location_id": 113,
                    "created_at": "2018-09-13 01:45:15",
                    "cross_street": null,
                    "address": null,
                    "address2": null,
                    "city": null,
                    "state": null,
                    "country": null,
                    "postal_code": null,
                    "cc": null,
                    "latitude": null,
                    "longitude": null,
                    "gender_name": null,
                    "gender_display_name": null,
                    "skill_level_name": null,
                    "skill_level_display_name": null,
                    "age_group_name": null,
                    "age_group_display_name": null,
                    "distance_miles": null
                }
            }
        ],
        "users_alerting": []
    }
}
```


Organizations

Organizations are a new feature of the platform server. They allow private (or public) groups, with members, posters, and administrators. This allows people to form teams, or trainer classes, with private events. It also allows organizations to have open or closed enrollment, with content available exclusively to members, or with public sponsored content.

Any administrator can add or update organization events, allowing businesses to share management duties.

There is no user interface for this in the current Plai mobile app. Also, there is no billing system configured yet, and there are no rules about setting up organizations. Currently, any user can create and administer an organization. This feature will evolve over time, but the API server currently supports multiple business models via the privacy settings and authorization rules.

The organization endpoints all start with /api/organizations. For the initial release, the list of “organizations” in the database will be simply be shown under the “Featured Locations” tab.

```
{
    "id": 1,
    "name": "Kiehn Inc",
    "description": "whiteboard visionary functionalities",
    "phone": "1-940-576-0727 x0812",
    "organization_email": "jacobson.kenton@example.com",
    "url": "http://www.prohaska.com/magni-et-vitae-minus-vitae-quibusdam.html",
    "twitter": "http://bednar.com/at-id-ratione-aliquid-dolorem",
    "instagram": "http://www.ebert.info/qui-assumenda-hic-fuga",
    "facebook": "http://www.ortiz.com/voluptas-fugiat-omnis-quis-sit-dignissimos-illum-dolor
    "image_url": "http://upton.com/",
    "video_url": "https://www.dach.com/iste-quia-quisquam-cum-ab-animi-itaque",
    "user_id": 3,
    "created_at": "2018-09-13 01:45:08",
    "updated_at": "2018-09-13 01:45:08",
    "cross_street": "Nestor Lane & Hallie Lock",
    "address": "936 Darrel Greens",
    "address2": "0",
    "city": "Port Isaacfurt",
    "state": "1",
    "country": "Mayotte",
    "postal_code": "37198",
    "cc": "MN",
    "latitude": "-88.370430",
    "longitude": "-125.723809",
    "organization_admin_count": 0,
    "organization_poster_count": 0,
    "organization_member_count": 0,
    "distance_miles": 6287.695971697031
}
```


Organization Permissions

Every event has an **organization_id** field. In the future, other platform content, such as blog posts or advertisements, will also have an **organization_id** field.

By default, organization_id is NULL. In this case, the default user permission rules will apply: the creator/owner can always see it, and the event permissions will be one of 'Just Me (Draft Edit)', 'Friends Only', or 'Everyone (General Public)'. These permissions are based on the field **post_status_id**.

However, if organization_id is not NULL, then the event is associated with a particular organization. In this case, the user permission rules will be completely ignored, and instead the organization rules will apply.

The organization rules for content have one of the following settings: 'Just Me (Draft Edit)', 'Administrators Only', 'Posting Members Only', 'Members Only',

'Everyone (General Public)'. For organization events, individual user friendships are ignored, and only organization membership is considered. These permissions are based on the field **organization_post_status_id**.

Only users who have the 'Administrator' or 'Poster' role can post or edit organization events (or other future content, like blog posts).

These roles and permissions are intended to provide flexible options for Plai revenue models. Businesses could choose to restrict the 'Administrator' role to a few users, such as the managers or trainers of a gym. Or a group could put everyone in the 'Poster' role, such as a soccer team where any team member is allowed to post, or a cycle club that is using Plai as a team communication forum.

Users who are a 'Member' can read organization events, assuming it has the 'Members Only' permission, but they may not post events. This could be used by businesses (or individual trainers) to provide exclusive broadcast-only content.

Organization Enrollment

Organizations set the default role of new users by setting the enrollment policy, with the field **enrollment_policy_id**. It will be one of the following:

- 'Closed: Members can only be added by the owner, or another administrator'
- 'Open Enrollment: Anyone can join and become a member'
- 'Public Forum: Anyone can join and post content to members'

The list of possible values for **enrollment_policy_id** can be obtained from the /api/context endpoint.

Searching

The endpoints of /api/events, /api/users, and /api/organizations all allow searching of the database.

There are three styles of search supported by the platform:

Word searches

Word searches are a free-text search, like a google search of the platform database. The text fields will be searched for the words given. The name of the REST argument is "q", for query.

Each of the words are searched independently, so searching for "soccer tennis" returns events which contain either "soccer" or "tennis", and you'll get a list of both types of events.

This is designed to be used by users typing into a "Search" box. Only text fields are searched, not datetime or integer fields.

Operator searches

Operator searches are filters which apply to specific fields of the data model. For example, to see all of the events created by the user with id 69, the query can set "user_id=69" in the REST request for /api/events.

Unlike the word searches, where any word can match any field, the operator searches are combined together. All the rules filter rules match, not just one. This allows applications to apply a wide range of filter rules, for example, searching for an specific event type (on the **activity_type_id** field), within a certain date range (on the **start_datetime** field).

There are two ways to specify an operator search. For the most simple case, where you just want to filter for a specific value, simply set that field name to the value, like user_id=69. In REST, that would look like GET https://server/api/events?user_id=69.

However, operator searches allow for more than just a specific value. You can use greater than, less than, text searches, and more. You can use any of the following operators in your query:

```
$operators = [
    '=', '<', '>', '<=', '>=', '<>', '!=', '<=>',
    'like', 'like binary', 'not like', 'ilike',
    '&', '|', '^', '<<', '>>',
    'rlike', 'regexp', 'not regexp',
    '~', '~*', '!~', '!~*', 'similar to',
    'not similar to', 'not ilike', '~~*', '!~~*',
];
```

This is why they are called "operator" searches: because you can use many kinds of operators, not just the equals operator. And you can combine them to search within ranges, or within subsets of results.

For the non-simple case (not just an "equals" value), or to apply multiple filters applied to a given field, the REST argument name is slightly different. Instead of just using the field name as the key, like this:

```
key=...
user_id=...
```

```
start_datetime=...

#
# For example:
#
key=value
user_id=69
start_datetime=2018-09-13 12:00:00
```

One must instead use the key name with brackets `[]` appended:

```
key[]=...
user_id[]=...
start_datetime[]=...
```

This tells the server that we are sending an array of values, not just a single value, for this field name.

Next, the values that you send must be grouped as ordered pairs of "operator, value" for this particular field name. You can specify as many operators as you like. For example:

```
#
# Add the first filter condition:
#

# Less than operator:
start_datetime[]=<

# Less than this value:
start_datetime[]=2018-09-14 12:00:00

#
# Add a second filter condition:
#

# Greater than operator:
start_datetime[]=>

# Greater than this value:
start_datetime[]=2018-09-12 12:00:00

#
# Add a third filter, this time on a different field
#
```

```
# Equals operator (the 2nd '=')
user_id[]==

# Equals this value:
user_id[]=69
```

In a GET query string, this would look like:

‘” https://server/api/events?start_datetime[]=<&start_datetime[]=2018-09-14%2012:00:00&start_datetime[]=>&start_datetime[]=2018-09-12%2012:00:00&user_id[]=%60&user_id[]=69 ‘”

Note that the order of the arguments is important, as the (operator, value) pairs are parsed in order.

Combinations of filters can be used by client applications to create custom views. For example, they can display a specific calendar month of events, or all events of a certain type, or all future events from a particular organization, by filtering fields such as **start_datetime**, **organization_id**, and **activity_type_id**.

Searching dates and times

Doing a database search for datetimes can be tricky. If you simply try to search for the date “2018-09-12”, then the server will interpret that to mean the datetime of “2018-09-12 00:00:00”. It would only match items that have the time portion set to “00:00:00”. Similarly, you can’t just search for “2018-09” to get all events in September.

Therefor, the API has some helpers for searching on dates or datetimes. These helpers tell the server to use special SQL database syntax, so that you can more easily search for dates.

The date helpers are ‘day’, ‘month’, ‘year’, ‘time’, and ‘date’. To use the helper, just append it on to the end of your operator. For example, to find all events in 2018, use ‘=year’ instead of ‘=’. To search for events after a specific date, use ‘>date’ instead of ‘>’.

```
# Equals (year) operator:
start_datetime[]==year

# Equals this (year) value:
start_datetime[]=2018

# Greater than (date) operator:
start_datetime[]=>date
```

```
# Greater than this (date) value:
start_datetime[]=2018-09-13
```

Short field names (auto-complete)

Short field names are a convenience for developers. It's a server feature that allows for less typing, and shorter URLs.

When using operator searches, you might find that your REST request urls get very long and hard to read. For example, to search within a certain lat/lon box defined by four lat/lon corners, your REST GET request might look like this:

```
GET /api/events?latitude[]=>&latitude[]=69.0&latitude[]=<&latitude[]=70.0&longitude[]=>&long
```

Because of this, the server allows the use of auto-completed field names. So long as the name is an unambiguous prefix that matches exactly one field, you can use an abbreviated name. Instead of "start_datetime", you can just use "start", or even "sta", but not "s" because that would be ambigious with "skill_level". (Any ambiguous field names, like 's' or 'l', are ignored.)

In the above example, instead of latitude/longitude, you can use lat/lon:

```
GET /api/events?lat[]=>&lat[]=69.0&lat[]=<&lat[]=70.0&lon[]=>&lon[]=69.0&lon[]=<&lon[]=70.0
```

Built-in search filters: today_only, future_only, max_miles

Finally, in addition to the search paramers, some network endpoints have convenient pre-built filters.

GET /api/events has today_only=1 and future_only=1

```
GET /api/events?max_miles=1000
```

On events you can optionally set today_only=1 to get only events for today, both past and present, or future_only=1 to get only future events. This prevents the client app from needing to get today's date and construct a query for **start_datetime** manually.

FIXME both?

max_miles (with optional near_lat, near_lon)

Events, users, and organizations can also allow filtering by distance, using the **max_miles** option. For example, a REST request to

```
GET /api/events?max_miles=1000
```

would only show events within 1000 miles.

By default, the distance is calculated from the logged in user's profile latitude and longitude, which is considered their "home" value. But any latitude/longitude pair can be used for the distance location by passing the **near_lat** (for latitude) and **near_lon** (for longitude) arguments, and the server will dynamically calculate and filter the distance.

Paged / Chunked results

In the response that is returned from the server, the actual data will be returned under a JSON key name of "data". In addition to the actual "data" that we want, there will be two more keys that are dedicated to paging server results. These are called "links" and "meta":


```
{
    "data": [
        {
            "id": 8,
            ...snip...
            "distance_miles": 3034.491448586222
        }
    ],

    "links": {
        "first": "https://40.78.81.126/api/events?page=1",
        "last": "https://40.78.81.126/api/events?page=1",
        "prev": null,
        "next": null
    },
    "meta": {
        "current_page": 1,
        "from": 1,
        "last_page": 1,
        "path": "https://40.78.81.126/api/events",
        "per_page": 100,
        "to": 9,
```

```
        "total": 9
    }
}
```

These are used to page server results and chunk the data, instead of downloading huge query results in a single request. This is a part of the JSON API standard. It is documented here:

http://jsonapi.org/format/#fetching-pagination

Because this is a known standard, the REST/JSON library on your platform may offer functions or callbacks to automate data paging. (This can be used to create "infinitely scrolling" lists in a user interface, where visually scrolling to the bottom of the list automatically triggers the loading of the next page of data. This provides a peppy and efficient user interface experience.)

The /api/context Endpoint

There is one special endpoint that all clients will want to get when the user first logs in: /api/context .

It provides all the types, categories, roles, and permissions defined by the database. For each of these, there is an integer id, which should be used as the key when saving data on the server. For example, set activity_type_id=1 for a "Basketball" event, or activity_type_id=2 for a "Bike" event.

For each entry, there is also a "name". This is an all-lowercase, code-friendly variable name to be used for code logic. This is necessary so that source code doesn't have to refer to magic integer id numbers (like 1, 2, etc.) that might change in the future.

Finally, there is a "display_name", which is a human-friendly label. This may contain spaces, punctuation, etc.

Any user interface forms that create or edit data should use these server-based arrays as the select options, rather than hard-coding any list of possible select options. This will ensure that all id values are consistent across all platforms. It also allow us to add or remove values without releasing a new version of the client applications.

It is recommended that the client applications retrieve this once per login. For iPhone Swift, it could be stored in CoreData. For Android, it could be stored as part of the Activity Context, or a similar global singleton where your store application-wide data. For Vue this would be stored in Vuex.

```
{
    "data": {
        "activity_types": [
            {
                "id": 1,
                "name": "basketball",
                "display_name": "Basketball",
                "created_at": null,
                "updated_at": null,
                "deleted_at": null
            },
            {
                "id": 2,
                "name": "bike",
                "display_name": "Bike",
                "created_at": null,
                "updated_at": null,
                "deleted_at": null
            },
            {
                "id": 3,
                "name": "bowl",
                "display_name": "Bowl",
                "created_at": null,
                "updated_at": null,
                "deleted_at": null
            },
            {
                "id": 4,
                "name": "camping",
                "display_name": "Camping",
                "created_at": null,
                "updated_at": null,
                "deleted_at": null
            },
            {
                "id": 5,
                "name": "crossfit",
                "display_name": "Crossfit",
                "created_at": null,
                "updated_at": null,
                "deleted_at": null
            },
            {
                "id": 6,
                "name": "dance",
```

```
    "display_name": "Dance",
    "created_at": null,
    "updated_at": null,
    "deleted_at": null
},
{
    "id": 7,
    "name": "disc",
    "display_name": "Disc",
    "created_at": null,
    "updated_at": null,
    "deleted_at": null
},
{
    "id": 8,
    "name": "dog_walk",
    "display_name": "Dog Walk",
    "created_at": null,
    "updated_at": null,
    "deleted_at": null
},
{
    "id": 9,
    "name": "football",
    "display_name": "Football",
    "created_at": null,
    "updated_at": null,
    "deleted_at": null
},
{
    "id": 10,
    "name": "golf",
    "display_name": "Golf",
    "created_at": null,
    "updated_at": null,
    "deleted_at": null
},
{
    "id": 11,
    "name": "kickball",
    "display_name": "Kickball",
    "created_at": null,
    "updated_at": null,
    "deleted_at": null
},
{
```

```
    "id": 12,
    "name": "pilates",
    "display_name": "Pilates",
    "created_at": null,
    "updated_at": null,
    "deleted_at": null
},
{
    "id": 13,
    "name": "pingpong",
    "display_name": "Pingpong",
    "created_at": null,
    "updated_at": null,
    "deleted_at": null
},
{
    "id": 14,
    "name": "run",
    "display_name": "Run",
    "created_at": null,
    "updated_at": null,
    "deleted_at": null
},
{
    "id": 15,
    "name": "skate",
    "display_name": "Skate",
    "created_at": null,
    "updated_at": null,
    "deleted_at": null
},
{
    "id": 16,
    "name": "snow",
    "display_name": "Snow",
    "created_at": null,
    "updated_at": null,
    "deleted_at": null
},
{
    "id": 17,
    "name": "soccer",
    "display_name": "Soccer",
    "created_at": null,
    "updated_at": null,
    "deleted_at": null
```

```
},
{
    "id": 18,
    "name": "softball",
    "display_name": "Softball",
    "created_at": null,
    "updated_at": null,
    "deleted_at": null
},
{
    "id": 19,
    "name": "surf",
    "display_name": "Surf",
    "created_at": null,
    "updated_at": null,
    "deleted_at": null
},
{
    "id": 20,
    "name": "swim",
    "display_name": "Swim",
    "created_at": null,
    "updated_at": null,
    "deleted_at": null
},
{
    "id": 21,
    "name": "tennis",
    "display_name": "Tennis",
    "created_at": null,
    "updated_at": null,
    "deleted_at": null
},
{
    "id": 22,
    "name": "volleyball",
    "display_name": "Volleyball",
    "created_at": null,
    "updated_at": null,
    "deleted_at": null
},
{
    "id": 23,
    "name": "yoga",
    "display_name": "Yoga",
    "created_at": null,
```

```
            "updated_at": null,
            "deleted_at": null
        },
        {
            "id": 24,
            "name": "other",
            "display_name": "Other",
            "created_at": null,
            "updated_at": null,
            "deleted_at": null
        }
    ],
    "age_groups": [
        {
            "id": 1,
            "name": "any",
            "display_name": "Any",
            "created_at": null,
            "updated_at": null,
            "deleted_at": null
        },
        {
            "id": 2,
            "name": "adult",
            "display_name": "Adult",
            "created_at": null,
            "updated_at": null,
            "deleted_at": null
        },
        {
            "id": 3,
            "name": "senior",
            "display_name": "Senior",
            "created_at": null,
            "updated_at": null,
            "deleted_at": null
        },
        {
            "id": 4,
            "name": "youth",
            "display_name": "Youth",
            "created_at": null,
            "updated_at": null,
            "deleted_at": null
        }
    ],
```

```
"genders": [
    {
        "id": 1,
        "name": "m",
        "display_name": "Male",
        "created_at": null,
        "updated_at": null,
        "deleted_at": null
    },
    {
        "id": 2,
        "name": "f",
        "display_name": "Female",
        "created_at": null,
        "updated_at": null,
        "deleted_at": null
    },
    {
        "id": 3,
        "name": "coed",
        "display_name": "Co-Ed",
        "created_at": null,
        "updated_at": null,
        "deleted_at": null
    },
    {
        "id": 4,
        "name": "none",
        "display_name": "Unspecified",
        "created_at": null,
        "updated_at": null,
        "deleted_at": null
    }
],
"skill_levels": [
    {
        "id": 1,
        "name": "casual",
        "display_name": "Casual",
        "created_at": null,
        "updated_at": null,
        "deleted_at": null
    },
    {
        "id": 2,
        "name": "intermediate",
```

```
        "display_name": "Intermediate",
        "created_at": null,
        "updated_at": null,
        "deleted_at": null
    },
    {
        "id": 3,
        "name": "advanced",
        "display_name": "Advanced",
        "created_at": null,
        "updated_at": null,
        "deleted_at": null
    }
],
"post_statuses": [
    {
        "id": 1,
        "name": "unposted",
        "display_name": "Just Me (Draft Edit)",
        "created_at": null,
        "updated_at": null,
        "deleted_at": null
    },
    {
        "id": 2,
        "name": "friends",
        "display_name": "Friends Only",
        "created_at": null,
        "updated_at": null,
        "deleted_at": null
    },
    {
        "id": 3,
        "name": "public",
        "display_name": "Everyone (General Public)",
        "created_at": null,
        "updated_at": null,
        "deleted_at": null
    }
],
"organization_post_statuses": [
    {
        "id": 1,
        "name": "unposted",
        "display_name": "Unposted (Draft Edit)",
        "created_at": null,
```

```
            "updated_at": null,
            "deleted_at": null
        },
        {
            "id": 2,
            "name": "admins",
            "display_name": "Administrators Only",
            "created_at": null,
            "updated_at": null,
            "deleted_at": null
        },
        {
            "id": 3,
            "name": "posters",
            "display_name": "Posting Members Only",
            "created_at": null,
            "updated_at": null,
            "deleted_at": null
        },
        {
            "id": 4,
            "name": "members",
            "display_name": "Members Only",
            "created_at": null,
            "updated_at": null,
            "deleted_at": null
        },
        {
            "id": 5,
            "name": "public",
            "display_name": "Everyone (General Public)",
            "created_at": null,
            "updated_at": null,
            "deleted_at": null
        }
    ],
    "enrollment_policies": [
        {
            "id": 1,
            "name": "closed",
            "display_name": "Closed: Members can only be added by the owner, or another
            "created_at": null,
            "updated_at": null,
            "deleted_at": null
        },
        {
```

```
                "id": 2,
                "name": "open_for_members",
                "display_name": "Open Enrollment: Anyone can join and become a member",
                "created_at": null,
                "updated_at": null,
                "deleted_at": null
            },
            {
                "id": 3,
                "name": "open_for_posters",
                "display_name": "Public Forum: Anyone can join and post content to members",
                "created_at": null,
                "updated_at": null,
                "deleted_at": null
            }
        ]
    }
}
```


Plai API Reference

Application Context

Get Context (Lists of Types, Roles, Policies, Permission Levels, etc.)

`GET /api/context`

Description

This gets the list of activity_types, age_groups, genders, skill_levels, post_statuses, organization_post_statuses, and enrollment_policies. These values say what list of "select" or "multiple select" options to provide in input forms.

The id values here should be used for the input values of `activity_type_id`, `age_group_id`, `gender_id`, `skill_level_id`, `post_status_id`, `organization_post_status_id`, and `enrollment_policy_id` when creating or updating content.

See above for a recent example of the available values. (Ignore the created_at, updated_at, and deleted_at fields for this data.)

Authentication

Register as New Local User

`POST /api/auth/register`

Parameters

- **name**: The Plai username.
- **email**: The user's email address. This email address must not already exist in the database.
- **password**: The user's login password, which can be used at /api/auth/login

Optional Parameters

- **nickname**: An optional screen name.
- **avatar_url**: An optional URL to an avatar image. If this is not provided, an avatar image will be generated for the user on the Plai server.
- **avatar_file**: An optional file upload to use as the user's avatar.

Description

The provided email address must not already exist – email is the unique key used for identifying logins.

Login as a Local User

`POST /api/auth/login`

Parameters

- **email**: The user's email address, to identify them
- **password**: The user's login password. This can be changed using `PUT /api/users/{user}`

Description

The provided email address must not already exist.

Logout (invalidate the current Auth token)

`POST /api/auth/logout`

Refresh Auth Token

`POST /api/auth/refresh`

Description

This will invalidate the current auth token and return a new one, with a new expiration time.

Clients should POST to this url before the current token expires, in order to keep a login session alive.

Events

List or Search Events

`GET /api/events`

Parameters

- **future_only=1** to hide past events.
- **today_only=1** to limit events to those occurring today.
- **max_miles=NUMBER** to limit the events to those within max_miles. The location will use the requestor's profile latitude and longitude, unless **near_lat** and **near_lon** are also supplied in the request to override the default.

Supported Fields for Operator Searches

Any of the following field names can be used with operator searches.

```
$columns_to_search = [ 'title', 'description', 'url', 'phone', 'event_email',
    'twitter', 'instagram', 'facebook', 'image_url', 'video_url', 'organization_id',
    'start_datetime', 'end_datetime', 'post_status_id', 'organization_post_status_id',
    'user_id', 'gender_id', 'age_group_id', 'activity_type_id', 'skill_level_id',
    'cross_street', 'address', 'address2', 'city', 'state', 'country', 'postal_code',
    'cc', 'latitude', 'longitude' 'name' ];
```

Description

Event list, from oldest to newest.

Create Event

`POST /api/events`

Parameters

- **image_file**: An optional file upload for the image. The field `image_url` will be point to to the copy of the file on the server.

All the (non-computed, writable) fields in an Event record can be supplied.

The following fields are required:

```
'post_status_id',
'user_id',
'location_id',
```

```
'gender_id',
'age_group_id',
'activity_type_id',
'skill_level_id',
```

Description

Create an Event with the given values. The saved event object will be returned. Any user can create an event, but admin or poster permission is required if an **organization_id** is also supplied.

Event Detail

```
GET /api/events/{event}
```

Description

Get a specific event entry. Unlike the event list, this also includes the fields

```
"users_going": [],
"users_alerting": []
```

Which is the list of users attending (or being alerted) on this event, including their profiles (if their profile permissions allow).

Update Event

```
PUT /api/events/{event}
PATCH /api/events/{event}
```

Parameters

- **image_file**: An optional file upload for the image. The field `image_url` will be point to to the copy of the file on the server.

All the (non-computed, writable) fields in an Event record can be supplied.

Delete Event

`DELETE /api/events/{event}`

Event: User Going

`POST /api/events/{event}/going`

Description

Mark the authenticated user as going to the provided event. The event detail will be returned, which should now include the current user in the `users_going` array.

Event: User Not Going

`POST /api/events/{event}/notgoing`

Description

Mark the authenticated user as NOT going to the provided event. The event detail will be returned, which should not include the current user in the `users_going` array.

Event: User Alerting

`POST /api/events/{event}/alerting`

Description

Mark the authenticated user as wanting alerts on the provided event. The event detail will be returned, which should now include the current user in the `users_alerting` array.

Event: User Not Alerting

`POST /api/events/{event}/notalerting`

Description

Mark the authenticated user as NOT wanting alerts on the provided event. The event detail will be returned, which should not include the current user in the `users_alerting` array.

Users

> NOTE: Users are not created like other items, with `POST users/`. Instead user creation is handled by `/api/auth/register`, below.
>
> NOTE: Similary, there is no `DELETE users`. That must be handled seperately by the admin panel, as it is not a feature supported by the API.

List or Search Users

`GET /api/users`

Parameters

- **max_miles=NUMBER** to limit the organizations to those within max_miles. The location will use the requestor's profile latitude and longitude, unless **near_lat** and **near_lon** are also supplied in the request to override the default.

Supported Fields for Operator Searches

Any of the following field names can be used with operator searches.

```
$columns_to_search = [ 'name', 'user_id', 'gender_id',
                       'skill_level_id', 'age_group_id' ];
```

Description

User list, from newest to oldest (by created_by).

User Detail

`GET /api/users/{user}`

Description

Get a specific user entry. Unlike the user list, this also includes the fields


```
"organizations_where_admin": [],
"organizations_where_poster": [
    {
        "id": 1,
        "name": "Kiehn Inc",
        "description": "whiteboard visionary functionalities",
        "url": "http://www.prohaska.com/magni-et-vitae-minus-vitae-quibusdam.html",
        "phone": "1-940-576-0727 x0812",
        "organization_email": "jacobson.kenton@example.com",
        "twitter": "http://bednar.com/at-id-ratione-aliquid-dolorem",
        "instagram": "http://www.ebert.info/qui-assumenda-hic-fuga",
        "facebook": "http://www.ortiz.com/voluptas-fugiat-omnis-quis-sit-dignissimos-illum-c
        "image_url": "http://upton.com/",
        "video_url": "https://www.dach.com/iste-quia-quisquam-cum-ab-animi-itaque",
        "user_id": 3,
        "location_id": 101,
        "enrollment_policy_id": 3,
        "created_at": "2018-09-13 01:45:08",
        "updated_at": "2018-09-13 01:45:08",
        "deleted_at": null,
        "pivot": {
            "user_id": 51,
            "organization_id": 1
        }
    }
],
"organizations_where_member": []
```


In this example there is one entry in the `organizations_where_poster` list, but `organizations_where_admin` and `organizations_where_member` are empty.

User: Get Me

`GET /api/users/me`

Description

Get the User Detail record of the currently authenticated user. This is useful if don't know your own user.id but need it for a query, or want to display the user's profile.

User: Get Me

`GET /api/users/me`

Description

Get the User Detail record of the currently authenticated user. This is useful if need to know your own user.id a query, or want to display the user's profile.

Update User

`PUT /api/users/{user}`
`PATCH /api/users/{user}`

Parameters

- **image_file**: An optional file upload for the image. The field `image_url` will be point to to the copy of the file on the server.
- **avatar_file**: An optional file upload to use as the user's avatar.

All the (non-computed, writable) fields in an Organization record can be supplied.

Description

Get the User Detail record of the currently authenticated user. This is useful if don't know your own user.id but need it for a query, or want to display the user's profile.

Users: Friends

Users: Friends: Send a friend request (invite)

`POST /api/users/friends/requests/{recipient}`

Description

Invite another user to be a friend.

The {recipient} argument is a user.id integer key. (The sender is the authenticated user.) The result will be the currently pending friend request entry, which includes the `sender_id` (the authenticated user) and the `recipient_id`:


```
{
    "data": {
        "recipient_id": 1,
        "recipient_type": "App\\User",
        "status": 0,
        "sender_type": "App\\User",
        "sender_id": 52,
        "updated_at": "2018-09-14 04:57:34",
        "created_at": "2018-09-14 04:57:34",
        "id": 1
    }
}
```


If you try to friend someone more than once, you'll receive an error:


```
{
    "error": "Failed (perhaps it's a duplicate request?)"
}
```

Users: Friends: List pending friend requests

`GET /api/users/friends/requests/pending`

Description

Get the list of unanswered invites for the authenticated user.

Users: Friends: Accept a friend request

`POST /api/users/friends/requests/accept/{sender}`

Users: Friends: Deny a friend request

`POST /api/users/friends/requests/deny/{sender}`

Users: Friends: List denied friend requests

`GET /api/users/friends/requests/denied`

Users: Friends: Block a user

`POST /api/users/friends/block/{friend}`

Users: Friends: List blocked friend requests

`GET /api/users/friends/blocked`

Users: Friends: Unblock a user

`POST /api/users/friends/unblock/{friend}`

Users: Friends: List friends

`GET /api/users/friends`

Users: Friends: List friends of friends

`GET /api/users/friends_of_friends`

Users: Friends: List mutual friends with another user

`GET /api/users/mutual_friends/{other}`

Organizations

List or Search Organizations

`GET /api/organizations`

Parameters

- **max_miles=NUMBER** to limit the organizations to those within max_miles. The location will use the requestor's profile latitude and longitude, unless **near_lat** and **near_lon** are also supplied in the request to override the default.

Supported Fields for Operator Searches

Any of the following field names can be used with operator searches.

```
$columns_to_search = ['name', 'description', 'url', 'phone', 'organization_email',
    'twitter', 'instagram', 'facebook', 'image_url', 'video_url', 'user_id',
    'enrollment_policy_id', 'cross_street', 'address', 'address2', 'city', 'state',
    'country', 'postal_code', 'cc', 'latitude', 'longitude' ];
```

Description

Organization list, from closest to furthest (by lat/lon).

Create Organization

`POST /api/organizations`

Parameters

- **image_file**: An optional file upload for the image. The field `image_url` will be point to to the copy of the file on the server.

All the (non-computed, writable) fields in an Organization record can be supplied.

Description

Create an Organization with the given values. The saved organization object will be returned. As of 2018-09-13, any user can create an organization.

Organization Detail

`GET /api/organizations/{organization}`

Description

Get a specific organization entry. Unlike the organization list, this also includes the fields

```
"members": [],
"posters": [],
"admins": [],
"events": []
```

Which is the list of users in their roles (with profiles), and also events associated with this organization.

Update Organization

`PUT /api/organizations/{organization}`
`PATCH /api/organizations/{organization}`

Parameters

- **image_file**: An optional file upload for the image. The field `image_url` will be point to to the copy of the file on the server.

All the (non-computed, writable) fields in an Organization record can be supplied.

Description

Only the creator-owner can delete an organization. Administrators cannot delete it.

Delete Organization

`DELETE /api/organizations/{organization}`

Description

Only the creator-owner can delete an organization. Administrators cannot delete it.

Organization: Join

```
POST /api/organizations/{organization}/join/{user}
```

Description

Have the provided user join the organization with the role specified by the enrollment_policy_name. It assumes open enrollment.

The user id must be specified in the URL. A user can join themselves (and must provide their own user.id in the URL), or an organization administrator can join another user to their organization.

Organization: Leave

```
POST /api/organizations/{organization}/leave/{user}
```

Description

Have the provided user leave the organization, regardless of what role they have.

The user id must be specified in the URL. A user can leave themselves (and must provide their own user.id in the URL), or an organization administrator can remove another user from their organization.

Organization: Members, Posters, and Administrators

```
# Members:
GET /api/organizations/{organization}/members
POST /api/organizations/{organization}/members/{user}
DELETE /api/organizations/{organization}/members/{user}
```

```
# Posters:
GET /api/organizations/{organization}/posters
POST /api/organizations/{organization}/posters/{user}
DELETE /api/organizations/{organization}/posters/{user}

# Administrators:
GET /api/organizations/{organization}/admins
POST /api/organizations/{organization}/admins/{user}
DELETE /api/organizations/{organization}/admins/{user}
```

Description

An organization administrator (or its creator-owner) can assign other users to particular roles in the organization using `POST` to these endpoints.

When doing a GET for the roles, only a user summary is provided. Full user profiles are not included. For that, use `GET /api/organizations/{organization}`.

The DELETE methods here assume the user has the specified role. To remove a user from an organization regardless of their role, use `POST /api/organizations/{organization}/leave/{user}`.
