API Documentation - Walking Group Project

Table of Contents

Revision History

- Rev 1 (June 24): Initial version. Updated port number and added `hasFullData` field.
- Rev 2 (June 26): Clarified role of password in `POST /users/{id}`
- Rev 3 (July 3): Corrected `/login` to be through `POST`
- Rev 4 (July 7):
  - Updated API for accessing messages.
  - Updated most JSON data to reflect current fields.
  - Changed user's last GPS structure to store date as a string.
- Rev 5 (July 22): Added support for more permission requests.
- Rev 6 (July 30): Add delete group.

1. Getting Started

1.1 Overview

- All URLs in this document are relative to the server **https://cmpt276-1177-bf.cmpt.sfu.ca:8184**
  - so, the /getApiKey URL is actually: https://cmpt276-1177-bf.cmpt.sfu.ca:8184/getApiKey
- Accessing the server's functionality is done through URL "end-points" which expose certain information and respond to different HTTP requests (like GET, POST, or DELETE)
- All messages to the server, except for retrieving your team's API key, must include the API header.
- All messages should ask for JSON content type by including the header: `Content-Type: application/json`
- Experiment with the REST API using either the command-line tool `curl`, or the Chrome plug-in `Postman`. You will not be able to use a plain browser for most API end-points. See Piazza for the latest set of `curl` commands.

Incomplete Objects vs Full Data

- Objects returned by the server usually return all details about themselves, but often **abbreviate the objects they reference**. For example, a user has a list of groups they lead: when a user is returned from the server (in JSON), the groups listed in this array will be **reduced to their `ID` and `href`**; all other data of the group will not be included. This keeps the representation brief and prevents recursive linking of objects in JSON.
  - If you want the details on these incomplete objects, you may:
    - execute an additional server request to get the full details.
    - include the "JSON-DEPTH: 1" header to trigger the server replying with JSON objects expanded to one additional level.
  - Each object returned by the server tells you if it has full data or not. See its `hasFullData` property. If it's `true`, then that object has its fields completely filled in. If `false` then it likely only had the `ID` and `href` from the server.
    - If an object has full data, note that if one of its properties is itself an object, that object might have either full data or just `ID`/`href`. That sub-object has a `hasFullData` property to tell you which it is.
    - In this document, some examples don't show the `hasFullData` property to keep the size of the examples down.
- The server may support features which are not (yet?) requirements of the project. You may ignore any features that are not yet needed.

1.2 Errors

- Any errors detected by the REST API are returned using a consistent JSON format.
- For example, the result of trying to get information on a non-existent user is:
  - HTTP Status: 400 (Bad Request)
  - JSON Body:

```
{
  "timestamp" : 1519942175990,
  "status" : 400,
  "error" : "Bad Request",
  "exception" : "ca.cmpt276.walkinggroupserver.model.IdItemUtils.IdItemException",
  "message" : "Requested unknown user.",
  "path" : "/users/222"
}
```

2. API Key

2.1 Retrieve API Key: `/getApiKey`

`/getApiKey?groupName=`**`<Group Name>`**

Parameters

- **`<Group Name>`**: The name of your group, such as "`zucchini`" (without the quotes; case insensitive).

Notes

- `GET` request to URL to retrieve your group's API key.
- The API key identifies your group's application to the server so it can differentiate it from all other groups. Hence what your group does will not interfere with any other groups.
- All calls to other methods require you to pass your group's API key.
- To query the API key you only need to do this once, and can simply do it from a browser. Browse to (changing parameters as needed):
  `https://cmpt276-1177-bf.cmpt.sfu.ca:8184/getApiKey?groupName=zucchini`
- Curl command:
  `curl -k -s -i -H "Content-Type: application/json" -X GET "https://cmpt276-1177-bf.cmpt.sfu.ca:8184/getApiKey?groupName=zucchini"`

3. Login & Logout

3.1 Description: `POST /login`

Headers

- **`apiKey`**: Your group's API key.

Body

- Email and password of the user.

```
{
    "email": "Groot@sfu.ca",
    "password": "iAmGroot"
}
```

Returns

- HTTP status: 200 (OK)
- Return message has the Authorization header (highlighted below) filled in with the user's newly issued token.

```
HTTP/1.1 200
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
Strict-Transport-Security: max-age=31536000 ;
includeSubDomains
X-Frame-Options: DENY
Authorization: Bearer
eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJHcm9vdEBzZnUuY2EiLCJleHAiOjE1
MjA3OTY0NTJ9.pbgL8s8JvSwGpHqBoiufAqmdHy3kKRZjXI5qYfVoaSMenFAe
lLKQGM6AYU902AF2HjbTmmomZy6-j-BoS5Ydgw
Content-Length: 0
Date: Thu, 01 Mar 2018 19:27:32 GMT
```

Notes

- No authorization header is needed as this is the end-point which gives you the authorization token.
- If authorization fails (unknown user email, incorrect password) it will return HTTP 401 (Unauthorized).

3.2 Logout

- To log out, the client just discards its authorization token.
- There is (currently) no logout support on the server.
- The server uses JSON Web Tokens (JWT) for authentication. These tokens are encoded by the server to represent the credentials of the user and encode an expiry time. The server does not maintain a list of outstanding tokens, so it is unable to record that the token has logged out.

4. Users

4.1 Create User: `POST /users/signup`

Headers

- **`apiKey`**: Your group's API key.

Body

- The new user's information in JSON format:

```
{
   "name": "Mr. Unique",
   "email": "unique12@sfu.ca",
   "password": "iAmUnique",
   "birthYear": "2005",
   "birthMonth": "12",
   "address": "#1 big way, Surrey BC, H0H 0H0, Canada",
   "cellPhone": "+1.778.098.7765",
   "homePhone": "(604) 123-4567",
   "grade": "Kindergarten",
   "teacherName": "Mr. Big",
   "emergencyContactInfo": "Call my mom!"
}
```

- Required fields are email and password; others are optional.

Returns

- HTTP status: 201 (Created)
- Body: JSON for newly created user, including its filled in ID (used to uniquely identify users in further interactions).

```
{
  "id" : 1,
  "name" : "Mr. Unique",
  "email" : "unique12@sfu.ca",
  "birthYear" : 2005,
  "birthMonth" : 12,
  "address" : "#1 big way, Surrey BC, H0H 0H0, Canada",
  "cellPhone" : "+1.778.098.7765",
  "homePhone" : "(604) 123-4567",
  "grade" : "Kindergarten",
  "teacherName" : "Mr. Big",
  "emergencyContactInfo" : "Call my mom!",
  "monitoredByUsers" : [ ],
  "monitorsUsers" : [ ],
  "memberOfGroups" : [ ],
  "leadsGroups" : [ ],
  "lastGpsLocation" : {
    "lat" : null,
    "lng" : null,
    "timestamp" : null
  },
  "messages" : [ ],
  "currentPoints" : null,
  "totalPointsEarned" : null,
  "customJson" : null,
  "pendingPermissionRequests" : [ ],
  "hasFullData" : true,
  "href" : "/users/1"
}
```

Notes

- Each user must have unique email addresses.
- The password is stored on the server (hashed) and never returned via the REST API.

4.2 List Users: `GET /users`

Headers

- **`apiKey`**: Your group's API key.
- **`Authorization`**: "`Bearer <token>`", where <token> is the current user's token.

Body

- None

Returns

- HTTP status: 200 (OK)
- Body: JSON array of all users for your group.
- See next page; note some users have been removed from this output to keep it short (so some of the users it references are not found in this listing).

```
[ {
  "id" : 1,
  "name" : "Mr. Unique",
  "email" : "unique12@sfu.ca",
  "birthYear" : 2005,
  "birthMonth" : 12,
  "address" : "#1 big way, Surrey BC, H0H 0H0, Canada",
  "cellPhone" : "+1.778.098.7765",
  "homePhone" : "(604) 123-4567",
  "grade" : "Kindergarten",
  "teacherName" : "Mr. Big",
  "emergencyContactInfo" : "Call my mom!",
  "monitoredByUsers" : [ ],
  "monitorsUsers" : [ ],
  "memberOfGroups" : [ ],
  "leadsGroups" : [ ],
  "lastGpsLocation" : {
    "lat" : null,
    "lng" : null,
    "timestamp" : null
  },
  "messages" : [ ],
  "currentPoints" : null,
  "totalPointsEarned" : null,
  "customJson" : null,
  "pendingPermissionRequests" : [ ],
  "hasFullData" : true,
  "href" : "/users/1"
}, {
  "id" : 2,
  "name" : "Ms. Minimum Details",
  "email" : "minimum@test.com",
  "birthYear" : null,
  "birthMonth" : null,
  "address" : null,
  "cellPhone" : null,
  "homePhone" : null,
  "grade" : null,
  "teacherName" : null,
  "emergencyContactInfo" : null,
  "monitoredByUsers" : [ ],
  "monitorsUsers" : [ ],
  "memberOfGroups" : [ ],
  "leadsGroups" : [ ],
  "lastGpsLocation" : {
    "lat" : null,
    "lng" : null,
    "timestamp" : null
  },
  "messages" : [ ],
  "currentPoints" : null,
  "totalPointsEarned" : null,
  "customJson" : null,
  "pendingPermissionRequests" : [ ],
  "hasFullData" : true,
  "href" : "/users/2"
} ]
```


Notes

- **Objects referenced by each user object, such as those in the `monitorsUsers` field, do not display all their contents; they only display their ID and the URL.** Use that object's ID to retrieve all details on that specific user.

4.3 Get Single User by ID: `GET /users/{id}`

`{id}` is the ID number of the user you are retrieving

Headers

- **`apiKey`**: Your group's API key.
- **`Authorization`**: "`Bearer <token>`", where <token> is the current user's token.

Body

- None

Returns

- HTTP status: 200 (OK)
- Body: JSON for some object

```
{
  "id" : 1,
  "name" : "Mr. Unique",
  "email" : "unique12@sfu.ca",
  "birthYear" : 2005,
  "birthMonth" : 12,
  "address" : "#1 big way, Surrey BC, H0H 0H0, Canada",
  "cellPhone" : "+1.778.098.7765",
  "homePhone" : "(604) 123-4567",
  "grade" : "Kindergarten",
  "teacherName" : "Mr. Big",
  "emergencyContactInfo" : "Call my mom!",
  "monitoredByUsers" : [ ],
  "monitorsUsers" : [ ],
  "memberOfGroups" : [ ],
  "leadsGroups" : [ ],
  "lastGpsLocation" : {
    "lat" : null,
    "lng" : null,
    "timestamp" : null
  },
  "messages" : [ ],
  "currentPoints" : null,
  "totalPointsEarned" : null,
  "customJson" : null,
  "pendingPermissionRequests" : [ ],
  "hasFullData" : true,
  "href" : "/users/1"
}
```

Notes

- Arrays of objects referenced by the user objects, such as the `monitorsUsers` field, do not display all their contents; they only display their ID and the URL you could use to retrieve all details on that specific user.

4.4 Delete Single User: `DELETE /users/{id}`

`{id}` is the ID number of the user you are deleting

Headers

- **`apiKey`**: Your group's API key.
- **`Authorization`**: "`Bearer <token>`", where <token> is the current user's token.

Returns

- HTTP status: 204 (No Content)

Notes

- First removes the user from:
  - all monitoring relationships with other users,
  - being a member from any group, and
  - removes the user as leader of all groups.
- If the user was the leader of a group, the group will be left in the system but will no longer have a leader.

4.5 Edit User: `POST /users/{id}`

`{id}` is the ID number of the user you are editing

Headers

- **`apiKey`**: Your group's API key.
- **`Authorization`**: "`Bearer <token>`", where <token> is the current user's token.

Body

- The user's new information in JSON format:

```
{
    "name": "Mr. Unique - edited",
    "email": "unique12@sfu.ca",
    "birthYear": 1,
    "birthMonth": 2,
    "address": "Over the rainbow - edited",
    "cellPhone": "+1.778.098.7765 - edited",
    "homePhone": "(604) 123-4567 - edited",
    "grade": "Kindergarten - edited",
    "teacherName": "Mr. Big - edited",
    "emergencyContactInfo": "Call anyone! - edited",
    "currentPoints": null,
    "totalPointsEarned": null,
    "customJson": null,
    "monitoredByUsers" : [ {"id":2} ],
    "monitorsUsers" : [ {"id":2} ],
    "memberOfGroups" : [ {"id":2} ],
    "leadsGroups" : [ {"id":2} ],
    "href" : "/users/252",
    "id": 235,
    "lastGpsLocation": {
        "lat": 333.4567,
        "lng": 444.5422,
        "timestamp": "2033-04-23T18:25:43.511Z"
    },
    "password": "UNCHANGED"
}
```

- Almost all fields of the `User` are updated on server, even those you don't send. If you don't send a field, it will be initialized with an empty value. So send a full `User` object with all fields.
- It is therefore recommended that your application have at some point asked the server for the information on a specific user, then you edit that object, and finally transmit it back to the server for storage.
- The password is not stored by this command.
- Other fields which are ignored include `monitoredByUsers`, `monitorsUsers`, `memberOfGroups`, `leadsgroup`, `messages`, `id`, and `href`.
  - To set `monitoredByUsers`/`monitorsUsers`, see section on monitoring users.
  - To set fields related to group, see the group section of this document.
  - You are unable to change the `id`, as it is set by the server. Likewise `href` is read only.
  - *These fields are shown in gray highlight in the above listing.*

Returns

- HTTP status: 200 (OK)
- Body: JSON for the updated user, including its filled in ID (used to uniquely identify users in further interactions).

```
{
  "id" : 1,
  "name" : "Mr. Unique - edited",
  "email" : "unique12@sfu.ca",
  "birthYear" : 1,
  "birthMonth" : 2,
  "address" : "Over the rainbow - edited",
  "cellPhone" : "+1.778.098.7765 - edited",
  "homePhone" : "(604) 123-4567 - edited",
  "grade" : "Kindergarten - edited",
  "teacherName" : "Mr. Big - edited",
  "emergencyContactInfo" : "Call anyone! - edited",
  "monitoredByUsers" : [ ],
  "monitorsUsers" : [ ],
  "memberOfGroups" : [ ],
  "leadsGroups" : [ ],
  "lastGpsLocation" : {
    "lat" : null,
    "lng" : null,
    "timestamp" : null
  },
  "messages" : [ ],
  "currentPoints" : null,
  "totalPointsEarned" : null,
  "customJson" : null,
  "pendingPermissionRequests" : [ ],
  "hasFullData" : true,
  "href" : "/users/1"
}
```


Notes

- Each user must have unique email addresses. Trying to change the current user to an email address which is already in use by another user will generate an error.
- The password is stored on the server (hashed) and never returned via the REST API.

4.6 Get Single User by Email: `GET /users/byEmail`

`/users/byEmail?email=<email>`

Parameters

- **`<email>`**: Email address of the user you are retrieving (encode '`@`' as "`%40`")

Headers

- **`apiKey`**: Your group's API key.
- **`Authorization`**: "`Bearer <token>`", where <token> is the current user's token.

Body

- None

Returns

- HTTP status: 200 (OK)
- Body: JSON for some object

```
{
  "id" : 2,
  "name" : "Ms. Minimum Details",
  "email" : "minimum@test.com",
  "birthYear" : null,
  "birthMonth" : null,
  "address" : null,
  "cellPhone" : null,
  "homePhone" : null,
  "grade" : null,
  "teacherName" : null,
  "emergencyContactInfo" : null,
  "monitoredByUsers" : [ ],
  "monitorsUsers" : [ ],
  "memberOfGroups" : [ ],
  "leadsGroups" : [ ],
  "lastGpsLocation" : {
    "lat" : null,
    "lng" : null,
    "timestamp" : null
  },
  "messages" : [ ],
  "currentPoints" : null,
  "totalPointsEarned" : null,
  "customJson" : null,
  "pendingPermissionRequests" : [ ],
  "hasFullData" : true,
  "href" : "/users/2"
}
```

Notes

- If using CURL or similar tool to generate requests, note that the argument `<email>` must be correctly encoded as a URL, which means you cannot place an '@' in the URL directly. Your command would look like:

```
curl -k -s -i -H "Content-Type: application/json" \
     -H "apiKey: th25tnhte-uth2tnhy-toehut23hnt5h2tnh" \
     -H "Authorization: Bearer eyJhbGciO..." \
     -X GET "${ADDR_276_SERVER}/users/byEmail?email=minimum%40test.com"
```

- Arrays of objects referenced by the user objects, such as the `monitorsUsers` field, do not display all their contents; they only display their ID and the URL you could use to retrieve all details on that specific user.

4.7 Get Last GPS Location: `GET /users/{id}/lastGpsLocation`

`{id}` is the ID number of the user whose location you want to retrieve.

Headers

- **`apiKey`**: Your group's API key.
- **`Authorization`**: "`Bearer <token>`", where <token> is the current user's token.

Returns

- HTTP status: 200 (OK)
- Body: JSON object for the last location of the user. May be all null values if no last value is set.
- Example body with valid data:

```
{
  "lat" : 123.4567,
  "lng" : 987.5422,
  "timestamp" : "2012-04-23T18:25:43"
}
```

- Example body with no data:

```
{
  "lat" : null,
  "lng" : null,
  "timestamp" : null
}
```

Notes

- No HTTP body for the request.
- `timestamp` is a string.

4.8 Set Last GPS Location: `POST /users/{id}/lastGpsLocation`

`{id}` is the ID number of the user whose location you want to set.

Headers

- **`apiKey`**: Your group's API key.
- **`Authorization`**: "`Bearer <token>`", where <token> is the current user's token.

Body

- JSON object for GPS location of the user

```
{
    "lat": 123.4567,
    "lng": 987.5422,
    "timestamp": "2012-04-23T18:25:43"
}
```

Returns

- HTTP status: 200 (OK)
- Body: JSON object for GPS location of the user just recorded. It should be the same values that were just sent to the server.

```
{
  "lat" : 123.4567,
  "lng" : 987.5422,
  "timestamp" : "2012-04-23T18:25:43"
}
```

Notes

- `timestamp` is a string, so you may express the date/time in any format your application desires.
- The server only stores one GPS location; each time you set a new GPS location the previous one is discarded.

5. User Monitoring

5.1 monitorsUsers vs monitoredByUsers

- A user can monitor any number of users, and be monitored by any number of users.
- If user A monitors user B, then:
  - A stores that it monitors B, and
  - B stores that it is monitored by B.
- For each change in who monitors whom, the server ensures that both involved users are updated by the server.
- This section shows the commands for "`monitorsUsers`"; however, the same commands exist for "`monitoredByUsers`" with the corresponding change in meaning and URL.

5.2 Get Who a User Monitors: `GET /users/{id}/monitorsUsers`

`{id}` is the ID number of the user you are retrieving information about.

Headers

- **`apiKey`**: Your group's API key.
- **`Authorization`**: "`Bearer <token>`", where <token> is the current user's token.

Body

- None

Returns

- HTTP status: 200 (OK)
- Body: The user monitors a set of other users. The body is a JSON array of this set of users. For example, '`GET /users/6/monitorsUsers`' returns the body (not undated to latest fields)

```
[ {
  "id" : 7,
  "name" : "Alice in Wonderland",
  "email" : "alice@sfu.ca",
  "monitoredByUsers" : [ {
    "id" : 6,
    "href" : "/users/6"
  } ],
  "monitorsUsers" : [ ],
  "memberOfGroups" : [ ],
  "leadsGroups" : [ ],
  "href" : "/users/7"
}, {
  "id" : 8,
  "name" : "Bob the Builder",
  "email" : "bob@sfu.ca",
  "monitoredByUsers" : [ {
    "id" : 6,
    "href" : "/users/6"
  } ],
  "monitorsUsers" : [ ],
  "memberOfGroups" : [ ],
  "leadsGroups" : [ ],
  "unreadMessages" : [ ],
  "readMessages" : [ ],
  "href" : "/users/8"
} ]
```

5.3 Make it so User Monitors Another User: `POST /users/{id}/monitorsUsers`

`{id}` is the ID number of the user you are modifying

Headers

- **`apiKey`**: Your group's API key.
- **`Authorization`**: "`Bearer <token>`", where <token> is the current user's token.

Body

- The ID of the user whom the user is going to monitor.

```
{
    "id": 2
}
```

  - The body may be the entire User object, or just the ID field (as shown above). Either will work.

Returns

- HTTP status: 201 (Created)
- Body: JSON for the array of users that this user is monitoring (it's "`monitorsUsers`" array).
- For example, when having user #1 start monitoring user #3, it returns the following
  - Note: user #1 was already monitoring user #2
  - Note: Some fields omitted to shorten the output.

```
[ {
  "id" : 2,
  "name" : "Ms. Minimum Details",
  "email" : "minimum@test.com",
  "monitoredByUsers" : [ {
    "id" : 1,
    "hasFullData" : false,
    "href" : "/users/1"
  } ],
  "monitorsUsers" : [ ],
  "memberOfGroups" : [ ],
  "leadsGroups" : [ ],
  "lastGpsLocation" : {
    "lat" : null,
    "lng" : null,
    "timestamp" : null
  },
  "messages" : [ ],
  "currentPoints" : null,
  "totalPointsEarned" : null,
  "customJson" : null,
  "pendingPermissionRequests" : [ ],
  "hasFullData" : true,
  "href" : "/users/2"
}, {
  "id" : 3,
  "name" : "Groot, just Groot",
  "email" : "Groot@sfu.ca",
  "monitoredByUsers" : [ {
    "id" : 1,
    "hasFullData" : false,
    "href" : "/users/1"
  } ],
  "monitorsUsers" : [ ],
  "memberOfGroups" : [ ],
  "leadsGroups" : [ ],
  "lastGpsLocation" : {
    "lat" : null,
    "lng" : null,
    "timestamp" : null
  },
  "messages" : [ ],
  "currentPoints" : null,
  "totalPointsEarned" : null,
  "customJson" : null,
  "pendingPermissionRequests" : [ ],
  "hasFullData" : true,
  "href" : "/users/3"
} ]
```


Notes

- For example, if user #10 is to monitor user #21, then
  `POST /users/10/monitorsUsers`
  with body `{"id": 21}`
- If user A monitors B, then the server also records in B that B is monitored-by A.

5.4 Stop Monitoring a User: `DELETE /users/{idA}/monitorsUsers/{idB}`

If user A monitors user B, and we want to end this relationship:
`{idA}` is the ID number of the user who monitors user B.
`{idB}` is the ID number of the user who is monitored by user A.

Headers

- **`apiKey`**: Your group's API key.
- **`Authorization`**: "`Bearer <token>`", where <token> is the current user's token.

Body

- None.

Returns

- HTTP status: 204 (OK, but No Content)
- Body: None

6. Groups

6.1 List Groups: `GET /groups`

Headers

- **`apiKey`**: Your group's API key.
- **`Authorization`**: "`Bearer <token>`", where <token> is the current user's token.

Returns

- HTTP status: 200 (OK)
- Body: JSON array of all groups

```
[ {
  "id" : 1,
  "groupDescription" : "The Minion Group",
  "routeLatArray" : [ ],
  "routeLngArray" : [ ],
  "leader" : {
    "id" : 1,
    "hasFullData" : false,
    "href" : "/users/1"
  },
  "memberUsers" : [ {
    "id" : 2,
    "hasFullData" : false,
    "href" : "/users/2"
  }, {
    "id" : 3,
    "hasFullData" : false,
    "href" : "/users/3"
  } ],
  "customJson" : null,
  "hasFullData" : true,
  "href" : "/groups/1"
}, {
  "id" : 2,
  "groupDescription" : "My Second Group",
  "routeLatArray" : [ ],
  "routeLngArray" : [ ],
  "leader" : {
    "id" : 1,
    "hasFullData" : false,
    "href" : "/users/1"
  },
  "memberUsers" : [ ],
  "customJson" : null,
  "hasFullData" : true,
  "href" : "/groups/2"
} ]
```

Notes

- No HTTP body in request.
- The `leader` field of the returned JSON object only includes the ID and the URL to access that user. Other fields of the user can be accessed using the `/users/{userId}` end point.

6.2 Create New Group: `POST /groups`

Headers

- **`apiKey`**: Your group's API key.
- **`Authorization`**: "`Bearer <token>`", where <token> is the current user's token.

Body

- The group to be created.
- No mandatory fields, but should likely include a `groupDescription` (string) and `leader` (specify the user ID of the leader of this group), but may also have anything which can be specified in the group update POST message such as GPS points..

```
{
    "groupDescription": "The Minion Group",
    "leader": {"id":1}
}
```

Returns

- HTTP status: 200 (OK)
- Body: JSON object of the group.

```
{
  "id" : 1,
  "groupDescription" : "The Minion Group",
  "routeLatArray" : [ ],
  "routeLngArray" : [ ],
  "leader" : {
    "id" : 1,
    "hasFullData" : false,
    "href" : "/users/1"
  },
  "memberUsers" : [ ],
  "customJson" : null,
  "hasFullData" : true,
  "href" : "/groups/1"
}
```

Notes

- The `leader` field of the returned JSON object only includes the ID and the URL to access that user. Other fields of the user can be accessed using the `/users/{userId}` end point.

6.3 Get Group Details: `GET /groups/{id}`

`{id}` is the ID number of the group you are getting.

Headers

- **`apiKey`**: Your group's API key.
- **`Authorization`**: "`Bearer <token>`", where <token> is the current user's token.

Returns

- HTTP status: 200 (OK)
- Body: JSON object of the group.

```
{
  "id" : 1,
  "groupDescription" : "The Minion Group",
  "routeLatArray" : [ ],
  "routeLngArray" : [ ],
  "leader" : {
    "id" : 1,
    "hasFullData" : false,
    "href" : "/users/1"
  },
  "memberUsers" : [ {
    "id" : 2,
    "hasFullData" : false,
    "href" : "/users/2"
  }, {
    "id" : 3,
    "hasFullData" : false,
    "href" : "/users/3"
  } ],
  "customJson" : null,
  "hasFullData" : true,
  "href" : "/groups/1"
}
```

Notes

- No body in request message.

6.4 Update Group Details: `POST /groups/{id}`

`{id}` is the ID number of the group you are changing.

Headers

- **`apiKey`**: Your group's API key.
- **`Authorization`**: "`Bearer <token>`", where <token> is the current user's token.

Body

- The new values to store for this group.
- To change the membership of the group, use `/groups/{id}/memberUsers/…` end point.

```
{
  "groupDescription": "Actually, we are evil",
    "leader": {
      "id": 2
    },
  "routeLatArray": [ 49.15523,  49.2352, 60.2532,  52.25232],
  "routeLngArray": [157.25322, 158.2532, 100.252, 100.25323],
}
```

- Any additional fields in body JSON are likely ignored (such as `id`, `memberUsers`, or `href`).

Returns

- HTTP status: 200 (OK)
- Body: JSON of the group (assuming group already had some members, as added via `POST` to `/groups/{id}/memberUsers`).

```
{
  "id" : 5,
  "groupDescription" : "Actually, we are evil",
  "routeLatArray" : [ 49.15523, 49.2352, 60.2532, 52.25232 ],
  "routeLngArray" : [ 157.25322, 158.2532, 100.252, 100.25323 ],
  "leader" : {
    "id" : 2,
    "href" : "/users/2"
  },
  "memberOfGroups" : [ {
    "id" : 4,
    "href" : "/groups/4"
  } ],
  "href" : "/groups/5"
}
```

Notes

- GPS coordinate arrays (`routeLatArray` and `routeLngArray`) are just arrays of doubles. Your app can send any number of values to be stored in these. For example, you may want to store two points in each, representing the start and end. Or, store five points each to indicate start, three intermediate way-points, and the end.

6.5 Delete Group: `DELETE /groups/{id}`

`{id}` is the ID number of the group to be deleted.

Headers

- **`apiKey`**: Your group's API key.
- **`Authorization`**: "`Bearer <token>`", where <token> is the current user's token.

Returns

- HTTP status: 204 (No Content)

Notes

- No body in the request message.
- No body in the response message.

6.6 Get Members of Group: `GET /groups/{id}/memberUsers`

`{id}` is the ID number of the group you are getting the users of.

Headers

- **`apiKey`**: Your group's API key.
- **`Authorization`**: "`Bearer <token>`", where <token> is the current user's token.

Returns

- HTTP status: 200 (OK)
- Body: JSON array of all **users** who are members for your group.
  - Note: Some fields have been omitted to shorten the output.

```
[ {
  "id" : 2,
  "name" : "Ms. Minimum Details",
  "email" : "minimum@test.com",
  "monitoredByUsers" : [ {
    "id" : 1,
    "hasFullData" : false,
    "href" : "/users/1"
  } ],
  "monitorsUsers" : [ ],
  "memberOfGroups" : [ {
    "id" : 1,
    "hasFullData" : false,
    "href" : "/groups/1"
  } ],
  "leadsGroups" : [ ],
  "lastGpsLocation" : {
    "lat" : null,
    "lng" : null,
    "timestamp" : null
  },
  "messages" : [ ],
  "pendingPermissionRequests" : [ ],
  "hasFullData" : true,
  "href" : "/users/2"
}, {
  "id" : 3,
  "name" : "Groot, just Groot",
  "email" : "Groot@sfu.ca",
  "monitoredByUsers" : [ {
    "id" : 1,
    "hasFullData" : false,
    "href" : "/users/1"
  } ],
  "monitorsUsers" : [ ],
  "memberOfGroups" : [ {
    "id" : 1,
    "hasFullData" : false,
    "href" : "/groups/1"
  } ],
  "leadsGroups" : [ ],
  "lastGpsLocation" : {
    "lat" : null,
    "lng" : null,
    "timestamp" : null
  },
  "messages" : [ ],
  "pendingPermissionRequests" : [ ],
  "hasFullData" : true,
  "href" : "/users/3"
} ]
```


Notes

- No body in request message.
- This returns the array of users which are members of the group. The leader of the group is stored in the `leader` field and is likely (though not necessarily) not a member of the group (i.e., the server does *not* enforce that the leader is not a member).

6.7 Add New Member of Group: `POST /groups/{id}/memberUsers`

`{id}` is the ID number of the group you are adding a member to.

Headers

- **`apiKey`**: Your group's API key.
- **`Authorization`**: "`Bearer <token>`", where <token> is the current user's token.

Body

- ID of the user who is being added as a member of the group.

```
{
  "id": 2
}
```

Returns

- HTTP status: 200 (OK)
- Body: JSON array of all **users** who are members for your group.

```
[ {
  "id" : 2,
  "name" : "Ms. Minimum Details",
  "email" : "minimum@test.com",
  "birthYear" : null,
  "birthMonth" : null,
  "address" : null,
  "cellPhone" : null,
  "homePhone" : null,
  "grade" : null,
  "teacherName" : null,
  "emergencyContactInfo" : null,
  "monitoredByUsers" : [ {
    "id" : 1,
    "hasFullData" : false,
    "href" : "/users/1"
  } ],
  "monitorsUsers" : [ ],
  "memberOfGroups" : [ {
    "id" : 1,
    "hasFullData" : false,
    "href" : "/groups/1"
  } ],
  "leadsGroups" : [ ],
  "lastGpsLocation" : {
    "lat" : null,
    "lng" : null,
    "timestamp" : null
  },
  "messages" : [ ],
  "currentPoints" : null,
  "totalPointsEarned" : null,
  "customJson" : null,
  "pendingPermissionRequests" : [ ],
  "hasFullData" : true,
  "href" : "/users/2"
} ]
```

Notes

- Fails if the user is already a member of the group, or if the user does not exist.

6.8 Remove Member from Group:

`DELETE /groups/{groupId}/memberUsers/{userId}`

`{groupId}` is the ID number of the group you are modifying.
`{userId}` is the ID of the user you are removing from the group

Headers

- **`apiKey`**: Your group's API key.
- **`Authorization`**: "`Bearer <token>`", where <token> is the current user's token.

Returns

- HTTP status: 204 (No content)

Notes

- No body.
- No contents of return HTTP message.
- May fail if `{userId}` is not initially a member of group `{groupId}`.

7. In-App Messaging Support

7.1 List messages: `GET /messages`

Usage Options (Query Strings)

Examples

Return all messages:

`GET /messages`

Return messages for user 85 which are unread:

`GET /messages?touser=85&status=unread`

Return messages for user 85 which are read:

`GET /messages?touser=85&status=read`

Return messages to user 85 which are unread and emergency:

`GET /messages?touser=85&status=unread&is-emergency=true`

Return messages with `is-emergency` flag set:

`GET /messages?is-emergency=true`

Return non-emergency messages:

`GET /messages?is-emergency=false`

Return messages to user 85:

`GET /messages?touser=85`

Explanation

- `status`: Return only messages which are either already read (`status=read`) or unread (`status=unread`).
- `touser`: Return messages for the indicated user (by user ID).
- `is-emergency`: Return messages flagged as an emergency (`is-emergency=true`), or return messages not flagged as an emergency (`is-emergency=false`).

Headers

- **`apiKey`**: Your group's API key.
- **`Authorization`**: "`Bearer <token>`", where <token> is the current user's token.

Returns

- HTTP status: 200 (OK)
- Body: JSON for the array of messages found for the query specified. See box on right.

Notes

- No body in HTTP request.
- Any of the objects inside the message (such as the `fromUser`, or `toUser`) only include the `ID` and `href` for that object. Further requests must be sent to retrieve that information.

```
[ {
  "id" : 1,
  "timestamp" : 1531027757852,
  "text" : "I hurt my leg!",
  "fromUser" : {
    "id" : 2,
    "hasFullData" : false,
    "href" : "/users/2"
  },
  "toUser" : {
    "id" : 1,
    "hasFullData" : false,
    "href" : "/users/1"
  },
  "read" : false,
  "emergency" : true,
  "href" : "/messages/1",
  "hasFullData" : true
}, {
  "id" : 2,
  "timestamp" : 1531027757852,
  "text" : "I hurt my leg!",
  "fromUser" : {
    "id" : 2,
    "hasFullData" : false,
    "href" : "/users/2"
  },
  "toUser" : {
    "id" : 3,
    "hasFullData" : false,
    "href" : "/users/3"
  },
  "read" : false,
  "emergency" : true,
  "href" : "/messages/2",
  "hasFullData" : true
} ]
```

7.2 New message to group: `POST /messages/togroup/{groupId}`

`{groupId}` is the ID number of the group to which you are sending the message.

Headers

- **`apiKey`**: Your group's API key.
- **`Authorization`**: "`Bearer <token>`", where <token> is the current user's token.

Body

- The body of the new message.

```
{
    "text": "Cannot lead group today; I have a hole in my sock.",
    "emergency": false
}
```

Returns

- HTTP status: 201 (Created)
- Body: JSON ***array*** for the messages which were sent.

```
[ {
  "id" : 3,
  "timestamp" : 1531027817942,
  "text" : "Cannot lead group today; I have a hole in my sock.",
  "fromUser" : {
    "id" : 2,
    "hasFullData" : false,
    "href" : "/users/2"
  },
  "toUser" : {
    "id" : 1,
    "hasFullData" : false,
    "href" : "/users/1"
  },
  "read" : false,
  "emergency" : false,
  "href" : "/messages/3",
  "hasFullData" : true
}, {
  "id" : 4,
  "timestamp" : 1531027817942,
  "text" : "Cannot lead group today; I have a hole in my sock.",
  "fromUser" : {
    "id" : 2,
    "hasFullData" : false,
    "href" : "/users/2"
  },
  "toUser" : {
    "id" : 2,
    "hasFullData" : false,
    "href" : "/users/2"
  },
  "read" : false,
  "emergency" : false,
  "href" : "/messages/4",
  "hasFullData" : true
}, {
  "id" : 5,
  "timestamp" : 1531027817942,
  "text" : "Cannot lead group today; I have a hole in my sock.",
  "fromUser" : {
    "id" : 2,
    "hasFullData" : false,
    "href" : "/users/2"
  },
  "toUser" : {
    "id" : 3,
    "hasFullData" : false,
    "href" : "/users/3"
  },
  "read" : false,
  "emergency" : false,
  "href" : "/messages/5",
  "hasFullData" : true
} ]
```


Notes

- The message in the body of the request is sent to the following users as an unread message:
  - group leader,
  - each member of the group,
  - all users who monitor a member of the group (“parents”).
- The message will always be from the user who is currently logged in (i.e., the user which was used to generate the token being transmitted with the API request).

7.3 New message to the 'parents' of a user:

`POST /messages/toparentsof/{userId}`

`{userId}` is the ID number of the user to whose 'parents' you are sending the message.

Headers

- **`apiKey`**: Your group's API key.
- **`Authorization`**: "`Bearer <token>`", where <token> is the current user's token.

Body

- The body of the new message.

```
{
    "text": "Mom, I hurt my leg!",
    "emergency": true
}
```

Returns

- HTTP status: 201 (Created)
- Body: JSON ***array*** for the messages which were sent.

```
[ {
  "id" : 1,
  "timestamp" : 1531027757852,
  "text" : "Mom, I hurt my leg!",
  "fromUser" : {
    "id" : 2,
    "hasFullData" : false,
    "href" : "/users/2"
  },
  "toUser" : {
    "id" : 1,
    "hasFullData" : false,
    "href" : "/users/1"
  },
  "read" : false,
  "emergency" : true,
  "href" : "/messages/1",
  "hasFullData" : true
}, {
  "id" : 2,
  "timestamp" : 1531027757852,
  "text" : "Mom, I hurt my leg!",
  "fromUser" : {
    "id" : 2,
    "hasFullData" : false,
    "href" : "/users/2"
  },
  "toUser" : {
    "id" : 3,
    "hasFullData" : false,
    "href" : "/users/3"
  },
  "read" : false,
  "emergency" : true,
  "href" : "/messages/2",
  "hasFullData" : true
} ]
```

Notes

- Delivers the message (as an unread message) to all of the following:
  - all users who are monitoring the given user (in the path), plus
  - all users who lead a group this user in.
- The message will be from the user who is currently logged in (i.e., the user which was used to generate the token being transmitted with the API request).
- It is likely that `{userId}` will be the same as the user logged in; however, it need not be and is therefore not enforced by the server.

7.4 Get one message: `GET /messages/{id}`

`{id}` is the ID number of the message you want to retrieve

Headers

- **`apiKey`**: Your group's API key.
- **`Authorization`**: "`Bearer <token>`", where <token> is the current user's token.

Returns

- HTTP status: 200 (OK)
- Body: JSON for the message. Note the objects referenced by the message are not returned with full data.

```
{
  "id" : 1,
  "timestamp" : 1531027757852,
  "text" : "Mom, I hurt my leg!",
  "fromUser" : {
    "id" : 2,
    "hasFullData" : false,
    "href" : "/users/2"
  },
  "toUser" : {
    "id" : 1,
    "hasFullData" : false,
    "href" : "/users/1"
  },
  "read" : false,
  "emergency" : true,
  "href" : "/messages/1",
  "hasFullData" : true
}
```

Notes

- No body to the HTTP request.

7.5 Delete message: `DELETE /messages/{id}`

`{id}` is the ID number of the message you want to delete

Headers

- **`apiKey`**: Your group's API key.
- **`Authorization`**: "`Bearer <token>`", where <token> is the current user's token.

Returns

- HTTP status: 204 (No contents)

Notes

- No body in HTTP request.
- No body in HTTP response.
- When the message is deleted, it is removed from the user to whom it was sent.

7.6 Mark message as read/unread by user:

`POST /messages/{messageId}/mark-read-or-unread`

`{messageId}` is the ID number of the message whose 'read' status is changed.

Headers

- **`apiKey`**: Your group's API key.
- **`Authorization`**: "`Bearer <token>`", where <token> is the current user's token.

Body

- The body is a simple `boolean`: true/false. No {…}, no quotes, no field name; just `true` or `false`.
- To make the message **read** send body:

```
true
```

- To make the message **unread** send body:

```
false
```

Returns

- HTTP status: 200 (OK)
- Body: JSON for the message. See sample returned JSON on the right.

```
{
  "id" : 1,
  "timestamp" : 1531027757852,
  "text" : "I hurt my leg!",
  "fromUser" : {
    "id" : 2,
    "hasFullData" : false,
    "href" : "/users/2"
  },
  "toUser" : {
    "id" : 1,
    "hasFullData" : false,
    "href" : "/users/1"
  },
  "read" : false,
  "emergency" : true,
  "href" : "/messages/1",
  "hasFullData" : true
}
```

8. Permission Requests

8.1 General

When enabled, the server generates “Permission Requests” for certain operations. Users can see which permission requests are pending for them to approve or deny. When a permission request is generated, the requested operation does not complete at the time of the initial API call, but happens later when the last needed permission is given from another user.

Operations which generate permission requests:

- **Start monitoring another user**; permissions required from:
  - user to be monitored,
  - one of the user’s current ‘parents’ (if any),
  - user to be monitoring.
- **Stop monitoring another user**; permissions required from:
  - user to be monitored,
  - one of the user’s current ‘parents’ (if any).
- **Creating a group with a leader**; permission required from:
  - new leader,
  - one of the new leader’s ‘parents’ (if any).
- **Changing the leader of a group**; permission required from:
  - old leader (if any)
  - new leader (if any; may be nobody),
  - one of the new leader’s ‘parents’ (if any).
- **Joining a walking group**; permission required from:
  - user joining the group,
  - one of the user’s parents (if any),
  - leader of the group (if any).
- **Leave a walking group**; permission required from:
  - user leaving the group,
  - one of the user’s parents (if any).

Each user object returned by the server includes a list of permission requests which are pending being approved/denied by that user.

When an API call is made to the server which would trigger an action which now requires permission, the server will create one permission request:

- A permission request stores:
  - who requested the action
  - what action is being requested (A wanting to lead group G, X joining group Y, etc).
  - status:
    - pending: if it is not yet accepted or rejected.
    - rejected: if any of the users who are asked to give permission reject the request.
    - accepted: if, for each set of users who need to grant permission, one user of that set grants permission.
  - which users were asked to give permission for the action,
  - which users have accepted or rejected the request
- A Permission record may have multiple sets of users needing to give permission. For example, if child A requests to lead group G, then:
  - G’s current leader (if any) will have to approve it
  - A will have to approve it (this approval given automatically if A requested the change)
  - one of A's 'parents' will have to approve it
- Each user can see a list of permission requests which for which they were asked to give

permission. Plus each user object contains a list of pending permission requests which they can accept or deny.

- Once one user has accepted or denied a request, no other users in that same user-set may accept or deny the request. Though, they may still see the request for their reference. If they try to accept or deny the request, nothing happens.
- In cases where nobody need be asked for permission (i.e., the current user, who initiated the action, is sufficient for granting all required permissions) then the server completes the requested change without generating a permission request or asking anyone (even the current user) for permission.

Hint: You can work with the permission request status by making of type `PermissionStatus`:

```
enum PermissionStatus {
    PENDING,
    APPROVED,
    DENIED
}
```

8.2 Testing Header

By default, the server does ***not*** enable permission requests. To turn them on, you must include the following header in every API request you make:

```
PERMISSIONS-ENABLED: true
```

If this header is missing or holds the value `false` (case insensitive) then that API request will not generate a permission request: it will just allow it to happen without any permissions. This is for testing and would not be found in a production system. If you are using the sample Retrofit library with the `ProxyBuilder`, you likely want to add some code to the `intercept()` method of the `AddHeaderInterceptor` such as:

```
builder.header("permissions-enabled", "true");
```

8.3 Example Exchange

1. App set to enable permissions by changing ProxyBuilder.java's intercept() function to include the line:
   `builder.header("permissions-enabled", "true");`
2. User creates two users: child and parent.
3. User establishes that parent monitors child.
4. User child tries to create a group and make themselves lead the group.
5. Server creates the group, but leaves the leader as null.
6. Server creates a permission request object, adding it in the list of pending requests for parent.
7. Parent logs in, see permission request and ether:
   Approves: Server records approval and since this approval is now sufficient it sets child to lead the group.
   Denies: Server records the denial and no change happens to the group leader.

8.4 Common Pitfalls

- Neglecting to enable permissions in the header.
- When a user requests to do an action which requires permission, they implicitly grant permission for the action because they initiated it. Hence, you'll never need to approve a permission request for an action which you initiated.
- Not having the user who's making the group be monitored by another user.
- If additional permission(s) are required, then the group will still be created just the leader will not be set.

8.5 List permission requests: `GET /permissions`

Usage Options (Query Strings)

Return all permission requests:
`GET /permissions`
Only return permission requests for user 5:
`GET /permissions?userId=5`
Only return permission requests for user 5 which are pending for him/her to approve/deny:
`GET /permissions?userId=5&statusForUser=PENDING`
Only return permission requests which relate to group 42
`GET /permissions?groupId=42`
Only return permission requests which have been denied:
`GET /permissions?status=DENIED`
Returning only those permission requests which match all the query string options:
`GET /permissions?status=APPROVED&groupId=42&userId=5`

- `groupId`: Return permission requests for the indicated group (by group ID). These are likely related to setting the leader of a group.
- `userId`: Return permission requests for the indicated user (by user ID). It searches the set of users which were asked to approve/deny the permission request for the given user ID.
  - with userId, may specify statusForUser as PENDING, APPROVED, or DENIED. For example, including PENDING returns only requests which are awaiting a response from the indicated user. Note that the list of PENDING requests for a specific user is included in each User object returned by the server.
- `status`: filters the permission requests based on the state of the entire permission request (not related to any single user).

Headers

- **`apiKey`**: Your group's API key.
- **`Authorization`**: "`Bearer <token>`", where <token> is the current user's token.

Returns

- HTTP status: 200 (OK)
- Body: JSON for the array of permission requests found for the query specified. See format of a single request on the next API call.

Notes

- No body in HTTP request.
- Any of the objects inside the permission request (such as `userA`) only include the `ID` and `href` for that object. Further requests must be sent to retrieve that information.

8.6 Get one permission request:

`GET /permissions/{id}`

`{id}` is the ID number of the permission request you want to retrieve

Headers

- **`apiKey`**: Your group's API key.
- **`Authorization`**: "`Bearer <token>`", where <token> is the current user's token.

Returns

- HTTP status: 200 (OK)
- Body: JSON for the permission request. Note the objects referenced by the message are returned in the reduced format of only listing the ID and the `href`.

Notes

- No body to HTTP request.
- The `action` field indicates the type of request. In this case that "User A" is to lead "Group G", which are both fields in the object.
- The `message` field describes the request, and is suitable for directly displaying to the user.
- Each of the "`authorizors`" is a group of users. Each group of users must have one of its users approve the request in order for it to be approved. When it is approved or denied, the user who did so is recorded in the `whoApprovedOrDenied` field.


```
{
   "id":14,
   "action":"A_LEAD_GROUP",
   "status":"APPROVED",
   "userA":{
      "id":105,
      "href":"/users/105"
   },
   "userB":null,
   "groupG":{
      "id":18,
      "href":"/groups/18"
   },
   "requestingUser":{
      "id":104,
      "href":"/users/104"
   },
   "authorizors":[
      {  "users":[
            {
               "id":108,
               "href":"/users/108"
            }
         ],
         "status":"APPROVED",
         "whoApprovedOrDenied":{
            "id":108,
            "href":"/users/108"
         }
      },
      {  "users":[
            {
               "id":107,
               "href":"/users/107"
            },
            {
               "id":106,
               "href":"/users/106"
            }
         ],
         "status":"APPROVED",
         "whoApprovedOrDenied":{
            "id":106,
            "href":"/users/106"
         }
      },
      {
         "users":[
            {
               "id":105,
               "href":"/users/105"
            }
         ],
         "status":"APPROVED",
         "whoApprovedOrDenied":{
            "id":105,
            "href":"/users/105"
         }
      }
   ],
   "message":"'Mr. Test User' (email:
testuser@test.com) asks that 'Little Pat'
(email: 3885@test.com) be allowed to begin
leading the group named 'Slow group'",
   "hasFulldata": true,
   "href":"/permissions/14"
}
```

8.7 Approve or deny a permission request:

`POST /permissions/{Id}`

`{Id}` is the ID number of the permission request to change.

Headers

- **`apiKey`**: Your group's API key.
- **`Authorization`**: "`Bearer <token>`", where <token> is the current user's token.

Body

- The body is a simple string: "APPROVED" or "DENIED"; quotes must be included around the string.

Returns

- HTTP status: 200 (OK)
- Body: JSON for the permission request.
- See sample returned for previous API call.

Notes

- Approves or denies it based on the currently logged in user.

8.8 Delete Permission: `DELETE /permissions/{id}`

`{id}` is the ID number of the permission to be deleted.

Headers

- **`apiKey`**: Your group's API key.
- **`Authorization`**: "`Bearer <token>`", where <token> is the current user's token.

Returns

- HTTP status: 204 (No Content)

Notes

- No body in the request message.
- No body in the response message.

9. Troubleshooting

- **If you get an HTTP 400 with no body:**
  This may mean:
  - you did not transmit to the API end-point the required information or format
  - your JSON object's syntax is incorrect (more likely with curl than Android)
    - Check for extra (missing) commas in your JSON object (more likely with curl)
  - your JSON object contains data of an incorrect format, like "hello" for a `long` etc.
  - your query string to the end-point is incorrect (argument names, data types)

  Ensure that you have provided the correct body contents, if required. Also, ensure that all fields in your JSON record exactly match the expected fields as a typo in your JSON field's name will cause a deserialization error on the server, resulting in a HTTP 400 response.

- If data you are sending to the server in a JSON message seems not to be saved, it may mean that you have a typo in the name of the JSON field (i.e., the name of your Java object being serialized). Ensure your name is an exact, case sensitive match to what the server expects.

- If you find a bug, please report it!