API Documentation - Walking Group Project

Table of Contents
1. Getting Started.................................................................................................................................................2
  1.1 Overview.....................................................................................................................................................2
  1.2 Errors..........................................................................................................................................................2
2. API Key.........................................................................................................................................................3
  2.1 Retrieve API Key: /getApiKey.................................................................................................................3
3. Login & Logout..................................................................................................................................................4
  3.1 Description: GET /login.........................................................................................................................4
  3.2 Logout.........................................................................................................................................................4
4. Users..............................................................................................................................................................5
  4.1 Create User: POST /users/signup............................................................................................................5
  4.2 List Users: GET /users.............................................................................................................................6
  4.3 Get Single User by ID: GET /users/{id}...................................................................................................7
  4.4 Delete Single User: DELETE /users/{id}..................................................................................................7
  4.5 Get Single User by Email: GET /users/byEmail.....................................................................................8
5. User Monitoring................................................................................................................................................9
  5.1 monitorsUsers vs monitoredByUsers........................................................................................................9
  5.2 Get Who a User Monitors: GET /users/{id}/monitorsUsers.....................................................................9
  5.3 Make it so User Monitors Another User: POST /users/{id}/monitorsUsers.............................................10
  5.4 Stop Monitoring a User: DELETE /users/{idA}/monitorsUsers/{idB}.....................................................11
6. Groups............................................................................................................................................................12
  6.1 List Groups: GET /groups.......................................................................................................................12
  6.2 Create New Group: POST /groups............................................................................................................13
  6.3 Get Group Details: GET /groups/{id}......................................................................................................14
  6.4 Update Group Details: POST /groups/{id}..............................................................................................15
  6.5 Delete Group: DELETE /groups/{id}.......................................................................................................16
  6.6 Get Members of Group: GET /groups/{id}/memberUsers........................................................................16
  6.7 Add New Member of Group: POST /groups/{id}/memberUsers.............................................................17
  6.8 Remove Member from Group: DELETE /groups/{groupId}/memberUsers/{userId}............................18

Revision History
- Rev 1 (March 1): Initial version (missing Groups)
- Rev 2 (March 4): Added /users/byEmail endpoint
- Rev 3 (March 9): Added /groups endpoints; updated user endpoints with group fields.
- Rev 4 (March 11): Added ability to delete a user.
1. Getting Started

1.1 Overview

- All URLs in this document are relative to the server https://cmpt276-1177-bf.cmpt.sfu.ca:8443, so, the /getApiKey URL is actually: https://cmpt276-1177-bf.cmpt.sfu.ca:8443/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.

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:

```json
{
    "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>&sfuUserId=<SFU User ID>

Parameters

- `<Group Name>`: The name of your group, such as “zucchini” (without the quotes; case insensitive).
- `<SFU User ID>`: Your SFU user ID, such as “bfraser” (without the quotes, case insensitive). Note that all members of the group will be given the same API key.

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:8443/getApiKey?
groupName=zucchini&sfuUserId=bfraser

- Curl command:
  
curl -k -s -i -H "Content-Type: application/json" -X GET "https://cmpt276-1177-bf.cmpt.sfu.ca:8443/getApiKey?
groupName=zucchini&sfuUserId=bfraser"
3. Login & Logout

3.1 Description: GET /login

Headers
- apiKey: Your group’s API key.

Body
- Email and password of the user.

```json
{
    "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 eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJHcm9vdEBzZnUyY2EiLCJleHAiOjE1MjA3OTY0NTJ9.pbgL8s8JvSwGpHqBoiufAqmdHy3kKR2jXISqYfVoaSMenFAe1LKQG6AYU902AF2HjbTmmom2y6-j-BoS5Ydqw
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:

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

Returns

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

```json
{
  "id" : 6,
  "name" : "Groot, just Groot",
  "email" : "Groot@sfu.ca",
  "monitoredByUsers" : [ ],
  "monitorsUsers" : [ ],
  "memberOfGroups" : [ ],
  "leadsGroups" : [ ],
  "href" : "/users/6"
}
```

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.

```json
[{
  "id" : 6,
  "name" : "Groot, just Groot",
  "email" : "Groot@sfu.ca",
  "monitoredByUsers" : [ ],
  "monitorsUsers" : [ { "id" : 7,
                      "href" : "/users/7"
                    } ],
  "memberOfGroups" : [ ],
  "leadsGroups" : [ ],
  "href" : "/users/6"
}, {
  "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" : [ ],
  "monitorsUsers" : [ ],
  "memberOfGroups" : [ ],
  "leadsGroups" : [ ],
  "href" : "/users/8"
}]
```

**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.
4.3 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" : 6,
  "name" : "Groot, just Groot",
  "email" : "Groot@sfu.ca",
  "monitoredByUsers" : [ ],
  "monitorsUsers" : [ {
    "id" : 7,
    "href" : "/users/7"
  } ],
  "memberOfGroups" : [ ],
  "leadsGroups" : [ ],
  "href" : "/users/6"
}
```

**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 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" : 6,
  "name" : "Groot, just Groot",
  "email" : "Groot@sfu.ca",
  "monitoredByUsers" : [ ],
  "monitorsUsers" : [ {
    "id" : 7,
    "href" : "/users/7"
  } ],
  "memberOfGroups" : [ ],
  "leadsGroups" : [ ],
  "href" : "/users/6"
}
```

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=bob%40sfu.ca"
```

- 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.
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 only shows the commands for “monitorsUsers”; however, the identical 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:

```
[ {
    "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" : [ ],
    "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.

```json
{
   "id": 8
}
```

**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 #6 start monitoring user #8, it returns the following (user #6 was already monitoring user #7):

```json
[
   {
      "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" : [ ],
      "href" : "/users/8"
   }
]
```

**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

```json
[ {
  "id" : 4,
  "groupDescription" : "Good Deeds Today",
  "routeLatArray" : [ ],
  "routeLngArray" : [ ],
  "leader" : {
    "id" : 1,
    "href" : "/users/1"
  },
  "memberUsers" : [ ],
  "href" : "/groups/4"
}, {
  "id" : 5,
  "groupDescription" : "The Minion Group",
  "routeLatArray" : [ ],
  "routeLngArray" : [ ],
  "leader" : {
    "id" : 1,
    "href" : "/users/1"
  },
  "memberUsers" : [ ],
  "href" : "/groups/5"
} ]
```

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": "Good Deeds Today",
    "leader": {"id":1}
}
```

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

```
{
    "id" : 4,
    "groupDescription" : "Good Deeds Today",
    "routeLatArray" : [ ],
    "routeLngArray" : [ ],
    "leader" : {
        "id" : 1,
        "href" : "/users/1"
    },
    "memberUsers" : [ ],
    "href" : "/groups/4"
}
```

**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.

```json
{
   "id" : 4,
   "groupDescription" : "Good Deeds Today",
   "routeLatArray" : [ ],
   "routeLngArray" : [ ],
   "leader" : {
      "id" : 1,
      "href" : "/users/1"
   },
   "memberOfGroups" : [ {
      "id" : 4,
      "href" : "/groups/4"
   } ],
   "href" : "/groups/4"
}
```

**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.

```json
[ {
   "id": 2,
   "name": "Little Minion",
   "email": "minionsRus@sfu.ca",
   "monitoredByUsers": [],
   "monitorsUsers": [],
   "memberOfGroups": [ { "id": 4,
                        "href": "/groups/4"
                       } ],
   "leadsGroups": [ { "id": 5,
                      "href": "/groups/5"
                     } ],
   "href": "/users/2"
} ]
```

**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 &lt;token&gt;”, where &lt;token&gt; is the current user’s token.

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

```json
{
   "id": 1
}
```

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

```json
[
   {
      "id" : 1,
      "name" : "Dr. Evil",
      "email" : "dr1evil@sfu.ca",
      "monitoredByUsers" : [ ],
      "monitorsUsers" : [ ],
      "memberOfGroups" : [ {
         "id" : 5,
         "href" : "/groups/5"
      } ],
      "leadsGroups" : [ {
         "id" : 4,
         "href" : "/groups/4"
      } ],
      "href" : "/users/1"
   }
]
```

**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}.