Walk With Talk

API Specification

Last Updated: March 27, 2015

On error, end-points will return a JSON string describing the error in the following format:

```
"error": "The error message",
"code":  200,   // An HTTP error code
"field": "The field that caused the error"
```

If the error is generic, that is, if the error is not restricted to a certain field (such as a duplicate username), all fields should be highlighted and the `"field"` key will contain `null`. A full list of HTTP error code and their uses can be found here.

The indication that a URL is `@authorized` means that it will return a 403 (forbidden) error unless there has previously been a successful login request sent.

`/URL`	Methods	Description

`/login`	`GET`, `POST`	Logs in a user.
		
		The `GET` action retrieves current user details. Results are returned in JSON format with the following fields:
		
		`"username",`
		`"email",`
		`"sex",`
		`"age"`
		
		The same fields are expected during the `POST` action, in addition to password data in the `"password"` field. This is a SHA-256 hash of the data entered by the user into the form field.
		
		The submission will return a 401 if the data does not match an existing user, with a `null` value for `"field"`. On success, there will be a code 200, as well as JSON data just like in the `GET` request.

`/register`	`POST`	Registers a new user. The `POST` action will expect the same fields as `/login`, in addition to a `"password_confirm"` field that follows the same characteristics as `"password"`. The submission will return a 409 if the username already exists, or if the passwords don't match. On success, there will the exact same value sent to the client as in the `/login` `GET` request.

`/schedule`
`@authorized`
	`POST`, `GET`	Updates / Retrieves the current user's schedule.
		
		The `POST` action expects the following fields in the following format:
		
		`"username",`
		`"monday",`
		`"tuesday",`
		`"wednesday",`
		`"thursday",`
		`"friday",`
		`"saturday",`
		`"sunday"`
		
		The username is self-explanatory, but the weekday fields require some more explanation. Each day is broken up into 30-minute discrete chunks. Each one of these is assigned a single bit, with a 1 indicating availability, and a 0 indicating the opposite. Thus, each of these fields should be a bit-string of length 48:
		
		$$\frac{\text{24 hour day} \cdot \text{60 minutes an hour}}{\text{30 minute chunks}} = 48 \text{ chunks}$$
		
		So an example schedule (for one day) could be
		
		`"monday": "000000110000011110001000010000011111100000000000"`
		
		The `GET` action will return the same thing outlined in the `POST` section, in JSON format. If the user doesn't have a schedule yet, all 0's will be substituted for every day.

`/location`
`@authorized`
	`POST`, `GET`	Updates / Retrieves the current user's location.
		
		The `POST` action expects the following fields:
		
		`"username",`
		`"latitude",`
		`"longitude"`
		
		Both of the coordinate fields should be floating-point values. See the `GET` action below for the return value.
		
		The `GET` action will return the most-recent location for the given user in the same format as the `POST` submission above, in JSON format. If no such location has been given yet, the coordinate fields will contain `null`.

`/nearby`	`GET`	Retrieves nearby "walkers" with matching schedules.
		
		The `GET` action takes no parameters, and returns a JSON array of possible users with the following information:
		
		`"nearby_users": {`
		`  "username_1" : 1.23, // Distance`
		`  "username_2" : 4.57`
		`}`
		
		No other information is provided about the users in order to ensure their privacy. Exact location information isn't provided, either, for the same reason. The front-end will have to make do with distances from the current user.

Handling "Instant" Scheduling

The user of the front-end client may want to find a walking buddy immediately, and there is no support for this directly through the API outlined above. An acceptable way of doing this would be to send four requests, as follows:

- `GET /schedule` and store the results.
- `POST /schedule` with only the current time block available
- `GET /nearby` to find everyone available right now
- `POST /schedule` to restore the original user schedule