**The Whoot API V1**

**Last updated: November 29, 2011**

**Notes**

- The production website is at [www.thewhoot.com](http://www.thewhoot.com), the staging website is at staging.thewhoot.com. Use the staging website for testing.

- All URLS (except for user search) are namespaced to /api/v1/{endpoint}

- All responses are JSON

- Always include the parameter ‘auth\_token’ to authenticate the user

- All response data is in the data property. Every response is in the following format:

{

‘status’:’ok|error|not\_authenticated’,

‘data’:{data}

}

- Form submissions with errors will respond with a 422 error code. The errors will be in the following format:

{

‘status’:’error’,

‘errors’:

[

{field}:’error’,

{field}:’error’

]

}

- \* indicates a required parameter

- We’re using the google maps API for the address/venue lookup field.

- Facebook App IDS:

Production – 224248554281682

Staging - 120786798012675

**POST ‘generate\_token’**

*This endpoint will generate a whoot user token that you can use on future requests to authenticate the user. If the user is not already a member of the whoot, it will create an account for them.* ***The app must first get a facebook session token for the user tied to our facebook app.***

**Params:**

**\***token - The Facebook session token

**Response:**

token – The Whoot user token

**GET ‘normal\_posts’**

*This endpoint will return an array of normal posts for the current user. It returns data representing the users main feed. You must provide a feed\_filters parameter. This parameter is a hash of options that dictates what types of feed items are returned and in what order.*

**Params:**

**\***feed\_filters – Hash in the form:

{

display:[‘working’, ‘low\_in’, ‘low\_out’, ‘big\_out’],

sort:{target:’created\_at’, order:’desc’}

}

*options for the above are:*

*display – working, low\_in, low\_out, big\_out*

*sort – created\_at, votes, comment\_count*

**Response:**

array of posts:

id

comment\_count

comments – array of

id

content

created\_at

created\_by

id

username

first\_name

last\_name

public\_id

vote\_count

voters – array of

id

username

first\_name

last\_name

public\_id

created\_at

night\_type

created\_by

id

first\_name

last\_name

email

public\_id

vote\_count

ping\_count

tag

id

name

venue

id

name

address

**POST ‘normal\_posts’**

*This endpoint will create a new normal post for the user.*

**Params:**

**\***normal\_post[night\_type] – (string) working | low\_in | low\_out | big\_out

normal\_post[tag][name] – (string) 40 char max, 3 word max

normal\_post[venue][name] – (string)

normal\_post[venue][address] – (string)

normal\_post[venue][coordinates\_string] – (string) {latitude}-{longitude}

**POST ‘normal\_posts/comments’**

*This endpoint will create a new comment on a post for the user.*

**Params:**

**\***comment[content] – (string)

\*comment[post\_id] – (string)

**POST ‘normal\_posts/votes’**

*This endpoint will create a new vote on a post for the user.*

**Params:**

**\***id – the post id

**POST ‘follow’**

*This endpoint will cause the current user to follow the target user.*

**Params:**

**\***id – the target user id (the user this user should follow)

**DELETE ‘follow’**

*This endpoint will cause the current user to stop following the target user.*

**Params:**

**\***id – the target user id (the user this user should follow)

**GET ‘users/:id/following’**

*This endpoint returns an array of users that the given user is following.*

**Params:**

**\***id – the target users PUBLIC id

**GET ‘users/:id/followers**

*This endpoint returns an array of users following the given user.*

**Params:**

**\***id – the target users PUBLIC id

**POST ‘ping’**

*This endpoint will cause the current user to ping the target user.*

**Params:**

**\***id – the target user id (the user this user should ping)

**GET ‘soul-data/search’**

*This endpoint will cause the current user to follow the target user.*

**Params:**

\*term: (string) the search term

\*limit: (int) limit number of results

\*types[]: ‘user’ AND/OR user\_id

If the user\_id is present the search service will return only users that the user\_id is following that match the term. If just ‘user’ is present as the type the search will return users matching the term. If both a user\_id and ‘user’ is present it will return two sets of results. One set will be users the user\_id is following that match the term and the other will be other users that match the term. There will be no duplicates.

*here is an example request:*

staging.thewhoot.com/soul-data/search?term=marc&limit=10&types[]=4ec1480acddc7f4de0000011&types[]=user

What this will do is return a set of results for users the 4ec1480acddc7f4de0000011 user is following that match ‘marc’ and general users that match ‘marc’.