Users

Create user:

Request:

HTTP POST: <server>/users/

Headers:

Content-Type: application/json (required)

X-foobar-username: <username>

X-foobar-access-token: <access token>

Request Body

{

"username": "rajaraodv",

"access\_token":"a123",

"account\_type": "facebook",

“first\_name”: “raja”,

“photo\_url”, “<http://facebook.com/photo.jpg>”

}

Response(s):

HTTP 200

{

**"createdAt"**:"2012-09-30T17:06:54.983Z",

**"updatedAt"**:"2012-09-30T17:06:54.983Z",

**"account\_type"**:"facebook",

**"username"**:"rajaraodv",

**"first\_name"**:"raja",

**"photo\_url"**:"http://facebook.com/photo.jpg",

**"\_id"**:"50687c2e9455fe0000000006"

}

HTTP 403

{

**"Error”**: "User with this username & access\_token already exists

}

HTTP 400 (Invalid Request - when content-type or something are improperly sent/not sent)

{  
 "Error": "invalid json. Tip: Check if headers (e.g. content-type), body(e.g. valid json) & query(e.g. encoding) parameters are all valid"  
}

**Notes / Tips for create user:**

1. Content-type application/json is required
2. You can send “username” and “access\_token” in POST body, or, in X-foobar-username and X-foobar-access-token headers. In general setting in header is preferred for most cases.
3. “account\_type” parameter is required and can be either “twitter” or “facebook”.

Get user:

Request:

HTTP GET: <server>/users/{id}

Example: <server>/users/50688fc9eadee00000000001

Headers:

~~Content-Type: application/json~~ **must not send**

X-foobar-username: <username> required

X-foobar-access-token: <access token> required

Response(s):

HTTP 200

{  
 "createdAt": "2012-09-30T18:30:33.961Z",  
 "updatedAt": "2012-09-30T18:30:33.961Z",  
 "account\_type": "facebook",  
 "username": "rajaraodv",  
 "last\_name": "rao",  
 "photo\_url": "http://facebook.com/photo.jpg",  
 "\_id": "50688fc9eadee00000000001"  
}

HTTP 400 (Invalid Request - when content-type or something are improperly sent/not sent)

{  
 "Error": "invalid json. Tip: Check if headers (e.g. content-type), body(e.g. valid json) & query(e.g. encoding) parameters are all valid"  
}

HTTP 404 (Not found - when requested user not found)

{  
 "Error": "User with id 50688fc9eadee00000000003 does not exist"  
}

HTTP 401 (Permission denied - Since everything is public, it’s usually when user w/ username + access\_token does not exist)

{  
 "Error": "User not authorized or does not exist"  
}

HTTP 400 (Invalid Request)

{  
 "Error": "Invalid Id. It must conform to MongoDB's objectId format"  
}

**Notes / Tips for get user:**

2. Must NOT send Content-type application/json header as there is no body-content.

4. X-foobar-username and X-foobar-access-token headers are Required.

5. id is required. id must be valid MongoDB’s default id(like 50688fc9eadee00000000001)

Update user:

Request:

HTTP PUT: <server>/users/{id}

Example: <server>/users/50688fc9eadee00000000001

Headers:

Content-Type: application/json (required)

X-foobar-username: <username> (required)

X-foobar-access-token: <access token> (required)

X-foobar-access-token-new: <new access token> (Optional - Only required (as header) if you want to update access\_token itself; Pass both access\_token and this one to update it)

Request Body

{

"username": "rajaraodv",

"account\_type": "facebook",

"last\_name": "new last name",

“photo\_url”, “<http://facebook.com/photo.jpg>”

}

Response(s):

HTTP 200

{

**"createdAt"**:"2012-09-30T17:06:54.983Z",

**"updatedAt"**:"2012-09-30T17:06:54.983Z",

**"account\_type"**:"facebook",

**"username"**:"rajaraodv",

**"last\_name"**:"rao",

**"photo\_url"**:"http://facebook.com/photo.jpg",

**"\_id"**:"50687c2e9455fe0000000006"

}

HTTP 400 (Invalid Request - when content-type or something are improperly sent/not sent)

{  
 "Error": "invalid json. Tip: Check if headers (e.g. content-type), body(e.g. valid json) & query(e.g. encoding) parameters are all valid"  
}

HTTP 400 (Invalid Request)

{  
 "Error": "Invalid Id. It must conform to MongoDB's objectId format"  
}

**Notes / Tips for update user:**

1. You can update ***currently logged in*** user only and not other users as there is no ACL.
2. {id} is required & must be currently logged in user’s MongoDB id.
3. Content-type application/json is required if you are updating something in the body.
4. To just update access\_token: set X-foobar-access-token-new header and send {} in the body

Delete user:

Request:

HTTP DELETE: <server>/users/{id}

Example: <server>/users/50688fc9eadee00000000001

Headers:

~~Content-Type: application/json~~ (Don’t send)

X-foobar-username: <username> (required)

X-foobar-access-token: <access token> (required)

Request Body

N/A

Response(s):

HTTP 200

{

**"status”: “OK”**

}

HTTP 400 (Invalid Request - when content-type or something are improperly sent/not sent)

{  
 "Error": "invalid json. Tip: Check if headers (e.g. content-type), body(e.g. valid json) & query(e.g. encoding) parameters are all valid"  
}

HTTP 400 (Invalid Request)

{  
 "Error": "Invalid Id. It must conform to MongoDB's objectId format"  
}

**Notes / Tips for delete user:**

1. You can delete ***currently logged in*** user only and not other users as there is no ACL.
2. {id} is required & must be currently logged in user’s MongoDB id.
3. Don’t send Content-type application/json
4. **TODO** - Currently we are not deleting user’s posts, comments and likes. Work in Progress.

Photo Posts (Individual Post w/ a Photo)

Create photoposts:

Request:

HTTP POST: <server>/photoposts/

Headers:

Content-Type: multipart/form-data (required)

X-foobar-username: <username>

X-foobar-access-token: <access token>

Request Body

**Note: Create a form w/ the following details:**

1. Upload Photo’s Field name must be: **“pic” (required)**
2. Field name for ‘username’ must be: **“username”** (required if X-foobar-username header is not set)
3. Field name for ‘access\_token’ must be: **“username”** (required if X-foobar-access-token header is not set)
4. Upload the photo to **/photoposts/** end point with enctype=**multipart/form-data**

Response(s):

HTTP 200

{  
 "\_\_v": 0,  
 "createdAt": "2012-10-01T01:58:05.323Z",  
 "updatedAt": "2012-10-01T01:58:05.323Z",  
 "creator": {  
 "\_id": "50687c2e9455fe0000000006",  
 "username": "rajaraodv",  
 "photo\_url": "http://facebook.com/photo.jpg"  
 },  
 "photo": {  
 "\_id": "5068f8ad28f5dc0000000001",  
 "url": "/photos/5068f8ad28f5dc0000000001",  
 "filename": "1-mainPhoto.png",  
 "width": "640",  
 "height": "960"  
 },

"\_id": "5068f8ad28f5dc0000000006",  
 "comments": [],  
 "comments\_cnt": 0,  
 "likes\_cnt": 0,

"photo\_caption": “” //Text user would add as a caption or description of the photo  
}

**Notes / Tips for creating photoposts:**

1. Please go through notes mentioned in Post-body section above
2. Notice that in response, you will get photo’s actual url as /photos/<photoId> (like /photos/5068f8ad28f5dc0000000001) and simply call that to retrieve photo.

Get PhotoPost (Returns detailed information of a given photopost - Could be used for Detailed-view)

Request:

HTTP POST: <server>/photoposts/{id}

Headers:

~~Content-Type: application/json~~ (Don’t send)

X-foobar-username: <username>

X-foobar-access-token: <access token>

Response(s):

HTTP 200

{  
 "\_id": "5068ed9f00c0f50000000008",  
 "creator": {  
 "photo\_url": "http://facebook.com/photo.jpg",  
 "username": "rajaraodv",  
 "\_id": "50687c2e9455fe0000000006"  
 },  
 "photo": {  
 "height": "960",  
 "width": "640",  
 "filename": "1-mainPhoto.png",  
 "url": "/photos/5068ed9f00c0f50000000003",  
 "\_id": "5068ed9f00c0f50000000003"  
 },  
 "updatedAt": "2012-10-01T01:24:43.994Z",  
 "comments": [  
 {  
 "updatedAt": "2012-10-01T01:13:04.108Z",  
 "creator": {  
 "photo\_url": "http://facebook.com/photo.jpg",  
 "username": "rrrr",  
 "\_id": "5068e8bea789370000000001"  
 },  
 "text": "some comment",  
 "\_id": "5068ee2000c0f50000000009"  
 },  
 {  
 "updatedAt": "2012-10-01T01:24:43.992Z",  
 "creator": {  
 "photo\_url": "http://facebook.com/photo.jpg",  
 "username": "rrrr",  
 "\_id": "5068e8bea789370000000001"  
 },  
 "text": "2nd Comment",  
 "\_id": "5068f0db01aeb10000000001"  
 }  
 ],  
 "comments\_cnt": 2,  
 "likes\_cnt": 0,

“photo\_caption”: “Golden Gate Bridge”

}

**Notes / Tips for creating photoposts:**

1. {id} is required. This is the id of post\_id.
2. This provides detailed info of a given photopost, including photos, all the comments, likes, etc.

Update PhotoPost: (Use this to add caption/description to photo)

Request:

HTTP PUT: <server>/photoposts/{id}

Headers:

Content-Type: application/json (required)

X-foobar-username: <username> (required)

X-foobar-access-token: <access token> (required)

Request Body

{

"photo\_caption": "This is photo of Golden Gate bridge",

}

Response(s):

HTTP 200

{  
 "\_\_v": 0,  
 "createdAt": "2012-10-01T01:58:05.323Z",  
 "updatedAt": "2012-10-01T01:58:05.323Z",  
 "creator": {  
 "\_id": "50687c2e9455fe0000000006",  
 "username": "rajaraodv",  
 "photo\_url": "http://facebook.com/photo.jpg"  
 },  
 "photo": {  
 "\_id": "5068f8ad28f5dc0000000001",  
 "url": "/photos/5068f8ad28f5dc0000000001",  
 "filename": "1-mainPhoto.png",  
 "width": "640",  
 "height": "960"  
 },

"\_id": "5068f8ad28f5dc0000000006",  
 "comments": [],  
 "comments\_cnt": 0,  
 "likes\_cnt": 0,

"photo\_caption": “This is photo of Golden Gate bridge”

}

HTTP 401

{

**"Error"**:"You can only update currently authenticated user's comment"

}

**Notes / Tips for updating comments:**

1. {id} is the \_id of the PhotoPost that’s being updated.
2. **photo\_caption** is required field.

Delete PhotoPost:

Request:

HTTP DELETE: <server>/photoposts/{id}

Example: <server>/photoposts/50688fc9eadee00000000001

Headers:

~~Content-Type: application/json~~ (Don’t send)

X-foobar-username: <username> (required)

X-foobar-access-token: <access token> (required)

Request Body

N/A

Response(s):

HTTP 200

{

**"status”: “OK”**

}

HTTP 400 (Invalid Request - when content-type or something are improperly sent/not sent)

{  
 "Error": "invalid json. Tip: Check if headers (e.g. content-type), body(e.g. valid json) & query(e.g. encoding) parameters are all valid"  
}

HTTP 400 (Invalid Request)

{  
 "Error": "Invalid Id. It must conform to MongoDB's objectId format"  
}

**Notes / Tips for delete user:**

1. This will delete PhotoPost, all associated Comments, all associated Likes and Photo itself.
2. You can only delete currently authenticated user’s PhotoPosts.

Get Photo (photo itself):

Request:

HTTP GET: <server>/photos/{id}

Example: <server>/photos/5068ed9f00c0f50000000003

Headers:

~~Content-Type: application/json~~ (Don’t send)

X-foobar-username: <username>

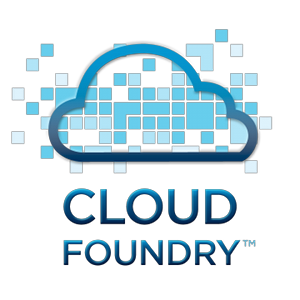
X-foobar-access-token: <access token>

Request Body

N/A

Response(s):

HTTP 200



Comments

Create comment:

Request:

HTTP POST: <server>/comments/

Headers:

Content-Type: application/json (required)

X-foobar-username: <username> (required)

X-foobar-access-token: <access token> (required)

Request Body

{

"post\_id": "5068ed9f00c0f50000000008",

"text": "2nd Comment"

}

Response(s):

HTTP 200

{  
 "createdAt": "2012-10-01T01:24:43.992Z",  
 "updatedAt": "2012-10-01T01:24:43.992Z",  
 "creator": {  
 "photo\_url": "http://facebook.com/photo.jpg",  
 "username": "rrrr",  
 "\_id": "5068e8bea789370000000001"  
 },  
 "text": "2nd Comment",  
 "post\_id": "5068ed9f00c0f50000000008",  
 "\_id": "5068f0db01aeb10000000001"  
}

HTTP 400

{  
 "Error": ‘Either ‘post\_id’ or ‘text’ parameter missing"  
}

HTTP 404

{  
 "Error": "PhotoPost Not Found"  
}

**Notes / Tips for create comments:**

1. Creating comment needs only ‘text’ field and ‘post\_id’ fields and both are **required**.
2. ‘text’ field is where you send comment’s text
3. ‘post\_id’ is the PhotoPost’s id to which this comment is meant for.
4. Note - When a comment is created, it’s parent PhotoPost’s updatedAt timestamp & comments\_cnt fields are also updated.
5. Todo - Currently there is no limit to comment’s length

Get Comment:

Request:

HTTP GET: <server>/comments/{id}

Example: <server>/photos/5068f0db01aeb10000000001

Headers:

~~Content-Type: application/json~~ (Don’t send)

X-foobar-username: <username>

X-foobar-access-token: <access token>

Request Body

N/A

Response(s):

HTTP 200

{  
 "createdAt": "2012-10-01T03:58:11.773Z",  
 "updatedAt": "2012-10-01T03:58:11.773Z",  
 "creator": {  
 "photo\_url": "http://facebook.com/photo.jpg",  
 "username": "rrrr",  
 "\_id": "5068fdb6b545cf0000000002"  
 },  
 "text": "2nd Comment",  
 "post\_id": "5068ed9f00c0f50000000008",  
 "\_id": "506914d3ea65ab0000000005"  
}

Update comment:

Request:

HTTP PUT: <server>/comments/{id}

Headers:

Content-Type: application/json (required)

X-foobar-username: <username> (required)

X-foobar-access-token: <access token> (required)

Request Body

{

"post\_id": "5068ed9f00c0f50000000008",

"text": " updated Comment"

}

Response(s):

HTTP 200

{  
 "createdAt": "2012-10-01T01:24:43.992Z",  
 "updatedAt": "2012-10-01T01:24:43.992Z",  
 "creator": {  
 "photo\_url": "http://facebook.com/photo.jpg",  
 "username": "rrrr",  
 "\_id": "5068e8bea789370000000001"  
 },  
 "text": "updated Comment",  
 "post\_id": "5068ed9f00c0f50000000008",  
 "\_id": "5068f0db01aeb10000000001"  
}

HTTP 401

{

**"Error"**:"You can only update currently authenticated user's comment"

}

**Notes / Tips for updating comments:**

1. {id} is \_id of the comment that’s being updated
2. You can only update ‘text’ field & is required.
3. Note - When you update a comment, in addition to the text, it’s updateAt timestamp & it’s parent PhotoPost’s updatedAt timestamp will also be updated.

Delete comment:

Request:

HTTP DELETE: <server>/comments/{id}

Example: <server>/comments/50688fc9eadee00000000001

Headers:

~~Content-Type: application/json~~ (Don’t send)

X-foobar-username: <username> (required)

X-foobar-access-token: <access token> (required)

Request Body

N/A

Response(s):

HTTP 200

{

**"status”: “OK”**

}

HTTP 400 (Invalid Request - when content-type or something are improperly sent/not sent)

{  
 "Error": "invalid json. Tip: Check if headers (e.g. content-type), body(e.g. valid json) & query(e.g. encoding) parameters are all valid"  
}

HTTP 400 (Invalid Request)

{  
 "Error": "Invalid Id. It must conform to MongoDB's objectId format"  
}

**Notes / Tips for delete user:**

1. You can only delete ***currently logged in*** user’s comment
2. {id} is required & must be currently logged in user’s MongoDB id.
3. Don’t send Content-type application/json

Likes

Create like:

Request:

HTTP POST: <server>/likes/

Headers:

Content-Type: application/json (required)

X-foobar-username: <username> (required)

X-foobar-access-token: <access token> (required)

Request Body

{

"post\_id": "5068ed9f00c0f50000000008"

}

Response(s):

HTTP 200

{  
 "\_\_v": 0,  
 "createdAt": "2012-10-02T00:56:01.216Z",  
 "updatedAt": "2012-10-02T00:56:01.216Z",  
 "creator": {  
 "\_id": "5069c1ed17f7eb0000000003",  
 "username": "rrrr",  
 "photo\_url": "http://facebook.com/photo.png"  
 },  
 "post\_id": "5069f0ddc2cdc00000000014",  
 "\_id": "506a3ba16a60a8f302000001"  
}

HTTP 403 (Forbidden)

{  
 "Error": "You can 'like' a post only once"  
}

HTTP 400

{  
 "Error": ‘post\_id’ parameter missing"  
}

HTTP 404

{  
 "Error": "PhotoPost Not Found"  
}

**Notes / Tips for create like:**

1. Creating comment needs only ‘post\_id’ field.
2. You can’t ‘like’ a post more than once, server will throw error.

Get Like:

Request:

HTTP GET: <server>/likes/{id}

Example: <server>/likes/5068f0db01aeb10000000001

Headers:

~~Content-Type: application/json~~ (Don’t send)

X-foobar-username: <username>

X-foobar-access-token: <access token>

Request Body

N/A

Response(s):

HTTP 200

{  
 "\_\_v": 0,  
 "createdAt": "2012-10-02T00:56:01.216Z",  
 "updatedAt": "2012-10-02T00:56:01.216Z",  
 "creator": {  
 "\_id": "5069c1ed17f7eb0000000003",  
 "username": "rrrr",  
 "photo\_url": "http://facebook.com/photo.png"  
 },  
 "post\_id": "5069f0ddc2cdc00000000014",  
 "\_id": "506a3ba16a60a8f302000001"  
}

Delete ‘Like’:

Request:

HTTP DELETE: <server>/likes/{id}

Example: <server>/likes/50688fc9eadee00000000001

Headers:

~~Content-Type: application/json~~ (Don’t send)

X-foobar-username: <username> (required)

X-foobar-access-token: <access token> (required)

Request Body

N/A

Response(s):

HTTP 200

{

**"status”: “OK”**

}

HTTP 400 (Invalid Request - when content-type or something are improperly sent/not sent)

{  
 "Error": "invalid json. Tip: Check if headers (e.g. content-type), body(e.g. valid json) & query(e.g. encoding) parameters are all valid"  
}

HTTP 400 (Invalid Request)

{  
 "Error": "Invalid Id. It must conform to MongoDB's objectId format"  
}

**Notes / Tips for delete user:**

1. You can only delete ***currently logged in*** user’s comment
2. {id} is required & must be currently logged in user’s MongoDB id.
3. Don’t send Content-type application/json

Feeds

Get like:

Request:

HTTP GET: <server>/feeds/${pageNumber}/${numberOfItems}

Example: <server>/feeds/1/10

Headers:

~~Content-Type: application/json (Don’t send)~~

X-foobar-username: <username> (required)

X-foobar-access-token: <access token> (required)

Request Body

N/A

Response(s):

HTTP 200

[  
 {  
 "\_id": "5069f0ddc2cdc00000000014",  
 "createdAt": "2012-10-01T19:37:01.593Z",  
 "creator": {  
 "photo\_url": "http://facebook.com/photo.png",  
 "username": "rajaraodv",  
 "\_id": "5069c1e617f7eb0000000002"  
 },  
 "photo": {  
 "height": "960",  
 "width": "640",  
 "filename": "1-mainPhoto.png",  
 "url": "/photos/5069f0ddc2cdc0000000000f",  
 "\_id": "5069f0ddc2cdc0000000000f"  
 },  
 "updatedAt": "2012-10-02T00:56:01.218Z",  
 "comments": [  
 {  
 "updatedAt": "2012-10-02T00:16:13.282Z",  
 "creator": {  
 "photo\_url": "http://facebook.com/photo.png",  
 "username": "rajaraodv",  
 "\_id": "5069c1e617f7eb0000000002"  
 },  
 "text": " from rajaraodv",  
 "\_id": "506a324d5d1dc00000000001"  
 },  
 {  
 "updatedAt": "2012-10-02T00:16:14.926Z",  
 "creator": {  
 "photo\_url": "http://facebook.com/photo.png",  
 "username": "rajaraodv",  
 "\_id": "5069c1e617f7eb0000000002"  
 },  
 "text": " from rajaraodv",  
 "\_id": "506a324e5d1dc00000000002"  
 }  
 ],  
 "comments\_cnt": 2,  
 "likes\_cnt": 1,  
 "photo\_caption": ""  
 },  
 {  
 "createdAt": "2012-10-01T21:52:58.584Z",  
 "updatedAt": "2012-10-01T21:52:58.584Z",  
 "creator": {  
 "photo\_url": "http://facebook.com/photo.png",  
 "username": "rajaraodv",  
 "\_id": "5069c1e617f7eb0000000002"  
 },  
 "photo": {  
 "height": "960",  
 "width": "640",  
 "filename": "1-mainPhoto.png",  
 "url": "/photos/506a10bae770810000000001",  
 "\_id": "506a10bae770810000000001"  
 },  
 "\_id": "506a10bae770810000000006",  
 "comments": [],  
 "comments\_cnt": 0,  
 "likes\_cnt": 0,  
 "photo\_caption": ""  
 }  
]

**Notes / Tips for create like:**

1. Returns an array of PhotoPosts.
2. ${pageNumber} - Required. Which page to return. Page number starts from 1.
3. ${numberOfItems} - Required. How many PhotoPost items to return (10 is typical).

Raw Schemas

(Use this to know required fields and other details)

User Schema

{

account\_type: {

type: String,

enum: accountTypes,

required: true

},

username: {

type: String,

required: true

},

first\_name: String,

last\_name: String,

access\_token: {

type: String,

required: true

},

account\_id: String,

photo\_url: {

type: String,

required: true

},

email: String

});

EmbeddedUser

//Simplified user schema for embedding in other objects like PhotoSchema

{

username: String,

\_id: Schema.Types.ObjectId,

photo\_url: String

}

Comment

{ creator: {

type: EmbeddedUserSchema.tree,

required: true

},

text: {

type: String,

required: true

},

post\_id: {

type: Schema.Types.ObjectId,

required: true

}

}

Like

{

creator: {

type: EmbeddedUserSchema.tree,

required: true

},

post\_id: {

type: Schema.Types.ObjectId,

required: true

}

}

Photo

{

url: {

type: String,

required: true

},

filename: {

type: String,

required: true

},

width: String,

height: String

}

PhotoPost

{

creator: {

type: EmbeddedUserSchema.tree,

required: true

},

photo: {

type: PhotoSchema.tree,

required: true

},

photo\_caption: {

type: String,

default:

""

},

likes\_cnt: {

type: Number,

default:

0

},

comments\_cnt: {

type: Number,

default:

0

},

comments: [{

type: Schema.Types.ObjectId,

ref: 'Comment'

}]

}