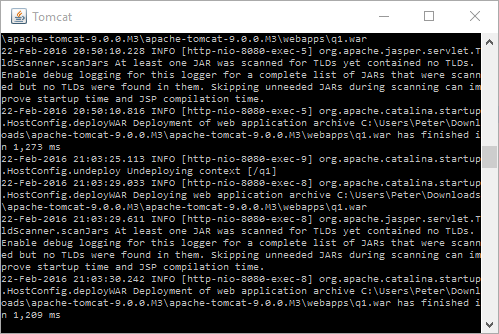
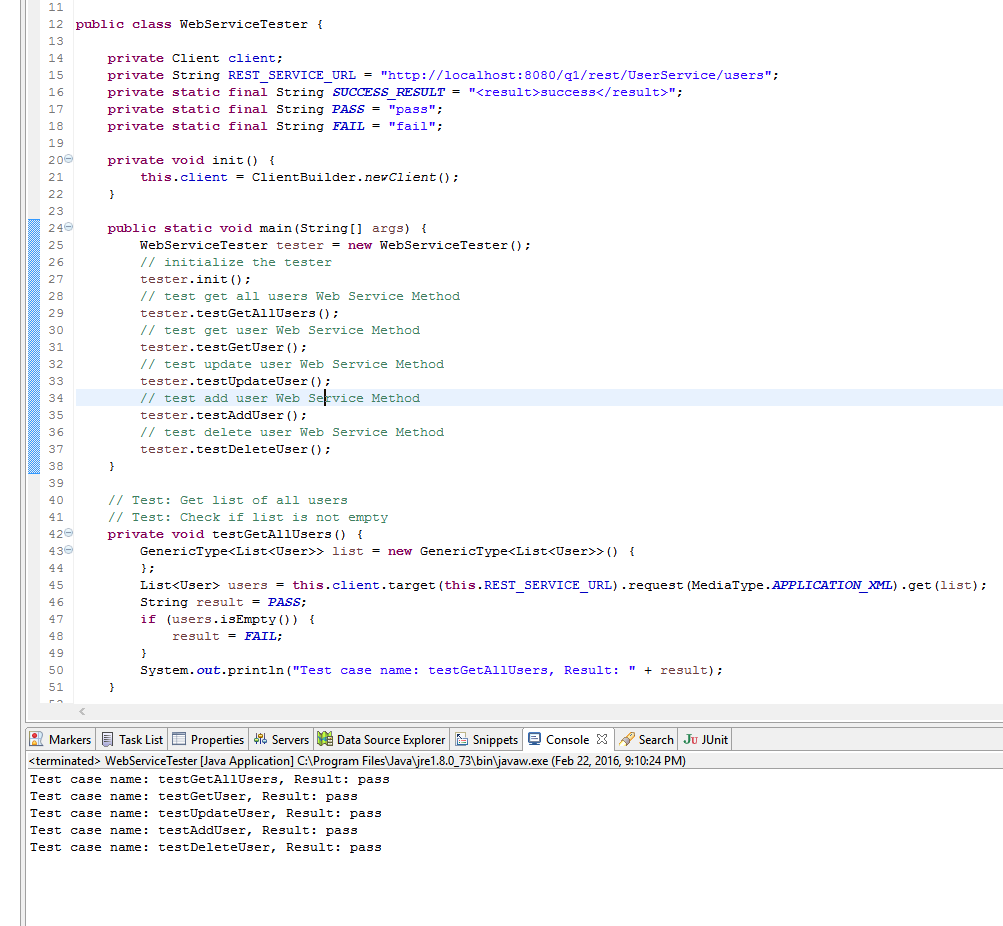
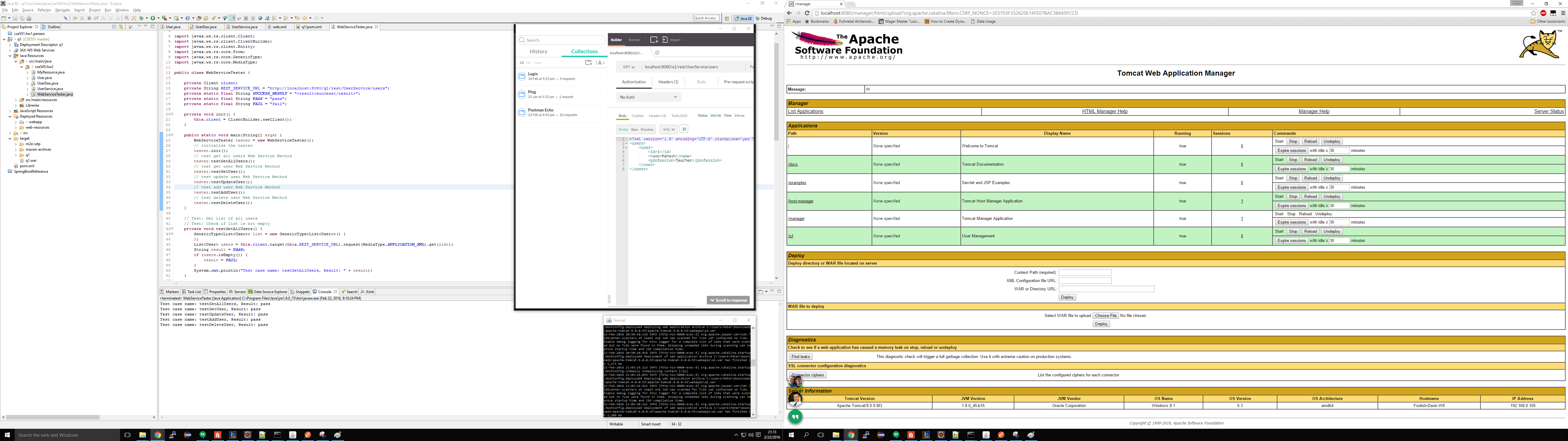
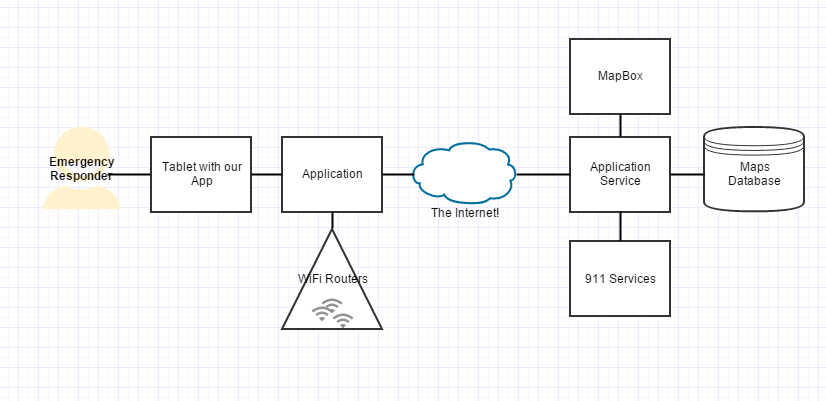
1)

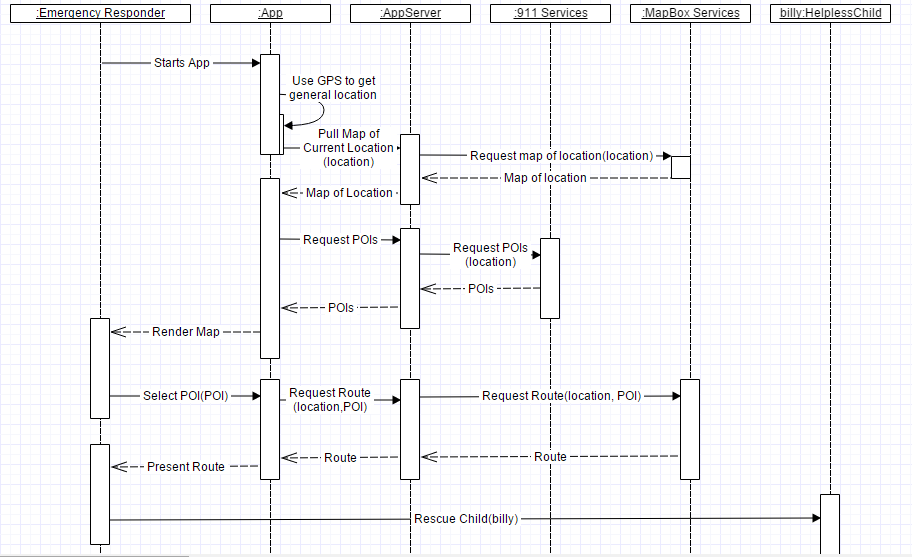
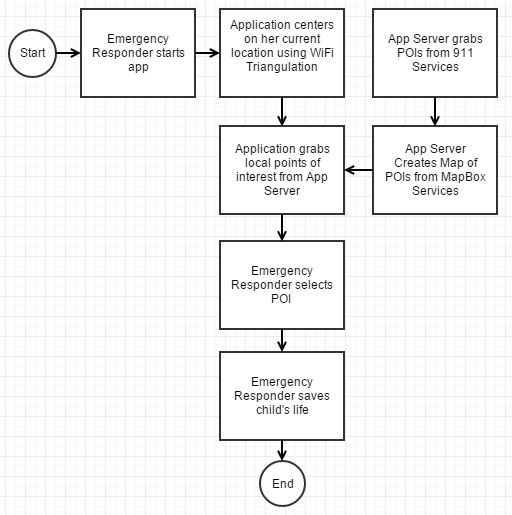


2)

To provide a means of indoor navigation to emergency personnel by utilizing already in place WiFi hotspots to determine current location. This can be accomplished by pre-mapping the building and its WiFi hotspots to determine WiFi signal strength at various locations. Then these locations can be mapped to latitude and longitude and paired with a floor of the building. The application can make a distinction on which floor the user is on by determining signal strength of known WiFi hotspots and display the desired map. Navigation is done by using a graph of nodes of locations connected by edges of viable paths. The nodes will likely occupy the same latitude and longitude in the building, however they will be connected between floors by using stairwell nodes. POIs can be a decoration on each location node to signify if the location has items of interest, such as chemicals, has someone currently on call with 911, or 911 has flagged the location as likely having individuals in need such as Alzheimer’s patients, disabled individuals, and small children.

* Rave Mobile Safety Smart911Connect API – Allows finding a caller within a corporate campus, might be able to expand to interior of building.
* MapBox – Rendering and interaction with map.
* Application MicroServices
  + Main Service
    - Holds and sends custom map data
  + Proxy Services
    - One for MapBox
    - One for 911 Services
  + Push Service
    - Able to send alerts and other immediate information to the user





3)

a)

The twitter API is RESTFUL but lacks anything but HTTP GET and HTTP POST due to what appears to be a desire to support HTML4 and XHTML1 only clients.

<http://stackoverflow.com/questions/165779/are-the-put-delete-head-etc-methods-available-in-most-web-browsers>

For images of our API documentation see appendix

|  |  |  |  |
| --- | --- | --- | --- |
| **API** | **Method** | **URI** | **Description** |
| Public | POST | https://api.twitter.com/1.1/statuses/update.json | Allows the user to post a status.  Returns a lot of information about the post such as its user's data for displaying profile pictures and who that user is for other things like mouse overs.  Returns data about how many retweets and such the post has, though as the post is brand new these values are likely always 0.  It has contains data about what hashtags are in the tweet, I'm not sure how this data is expected to be used.   Will deny duplicate requests with the same content:  {   "errors": [   {   "code": 187,   "message": "Status is a duplicate."   }   ]  } |
| Public | POST | https://api.twitter.com/1.1/statuses/destroy/:id.json | Allows the user to delete a status they own.   Returns the same data as update's initial creation. Essentially you get back what you put in on delete. Probably makes un-delete application level functionality easy.  When trying to delete another user's post it sends back:   "errors": [   {   "code": 183,   "message": "You may not delete another user's status."   } |
| Public | POST | https://api.twitter.com/1.1/statuses/retweet/693912311094976517.json | Retweets the tweet whose id you supply. Returns the original tweet with all the details (original poster, etc.) of the original tweet. |
| Public | GET | https://api.twitter.com/1.1/friends/list.json?user\_id=921229428 | Gives you back a detailed list of all the people that the given user\_id is following. |
| Public | POST | https://api.twitter.com/1.1/favorites/create.json?id=702976364660572161 | Before November 2015, the like action was known as favorite, so the API name still has the historic "favorites" name, although the new option is "like". It is an asynchronous process that lets create a like on a tweet. If the process is successful (200 OK response), it returns a "liked" status for the status' ID given as parameter of the authenticating user. Note: if try to favorite/like a tweet twice, the request will throw an error (from Postman). {   "errors": [   {   "code": 139,   "message": "You have already favorited this status."   }   ]  } |
| Public | GET | https://api.twitter.com/1.1/statuses/user\_timeline.json | The request returns a collection of the most recent tweets posted by the user (indicated by the screen\_name or user\_id parameters of the request). The timeline can only be requested if the authenticated user owns the timeline or is a follower of the user's timeline. The request can only return up to 3,200 recent user tweets in the timeline. |

b)

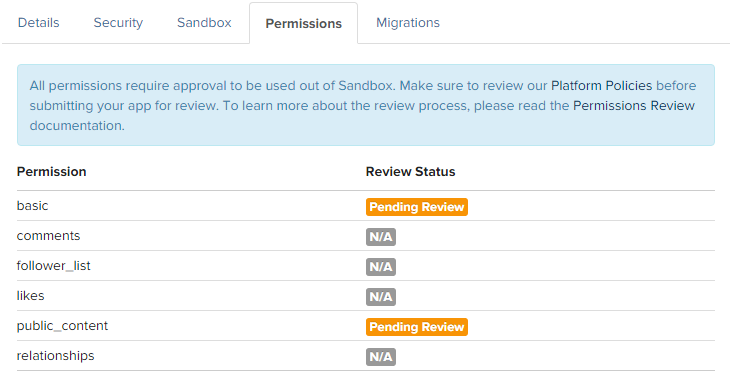
It appears that Instagram does not allow applications that aren’t major advertisers or content managers to access the public content permission for Instagram. This is after a API change from November 17th 2015. A common library for pulling a feed of publicly available Instagram pictures known and instafeed.js has an ongoing struggle to determine how it will continue going forward since all newly created API keys will not work for public content.

In the document you sent out it looks like Instagram declined your public content permission.

<http://www.carsonshold.com/2015/12/instagram-api-limits-valid-use-cases/>

<https://github.com/stevenschobert/instafeed.js>

Link with the discussion on how the api change breaks instafeed.js: <https://github.com/stevenschobert/instafeed.js/issues/345>

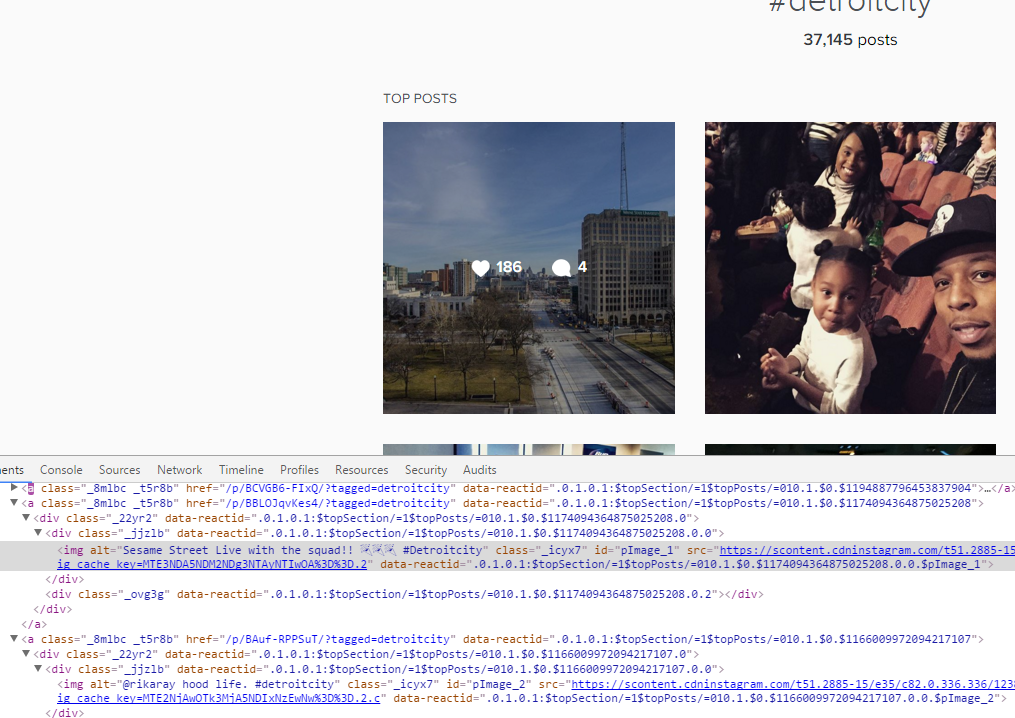


This looks like it is a continuation of the move by Facebook after purchasing Instagram to monetize Instagram and restrict access to content behind paywalls.

We potentially could write a web scraper that will pull the public tag search of Instagram to display images. This however seemed outside the scope of the question and would solve the problem but miss the point of the question.

It looks like explore/tags section of Instagram has an easy structure to parse with the images being stored in <img> tags with a class of “\_icyx7” so directly pulling the images shouldn’t be too complicated.

<https://www.instagram.com/explore/tags/detroitcity/>



4)

Result of example:



The request made by the example is meant for use by server applications and not JavaScript on a web page. The resource lacks an Access-Control-Allow-Origin header meaning JavaScript from another domain can not access the resource. This is documented in the API for the Places service.

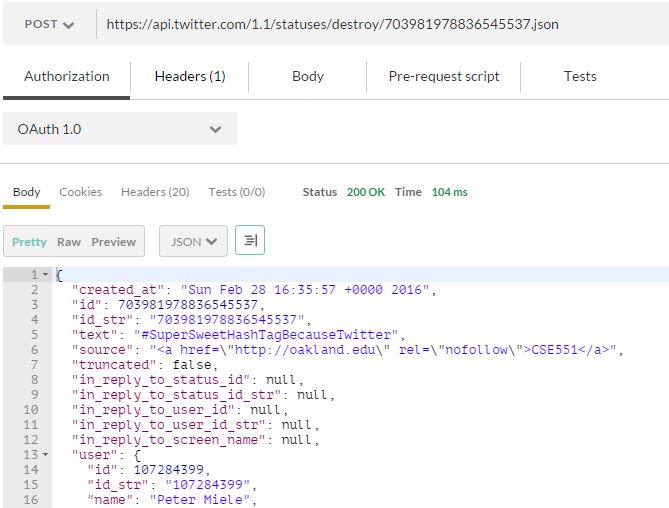
<https://developers.google.com/places/web-service/search>

It is suggested to use the Google Places API Javascript Library for these calls, which abstracts away all the XMLHttpRequest object logic and does all the work for you. Not very useful for an assignment about AJAX.

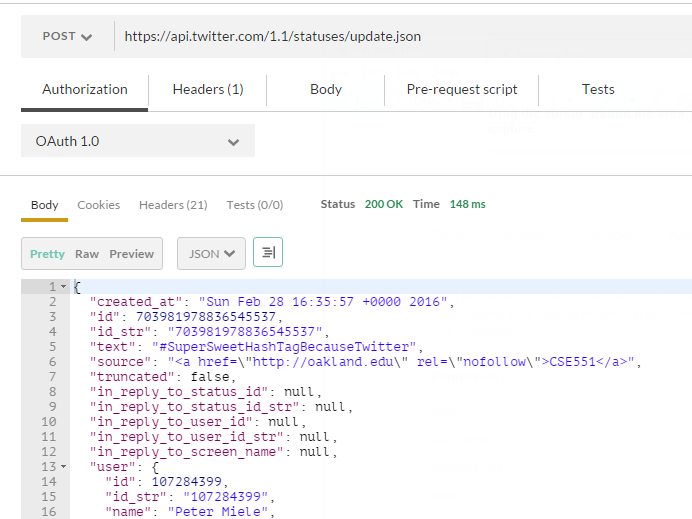
This could be done by writing a node.js middle-man to facilitate the request to Google’s places API for the desired data of national parks.

Appendix:

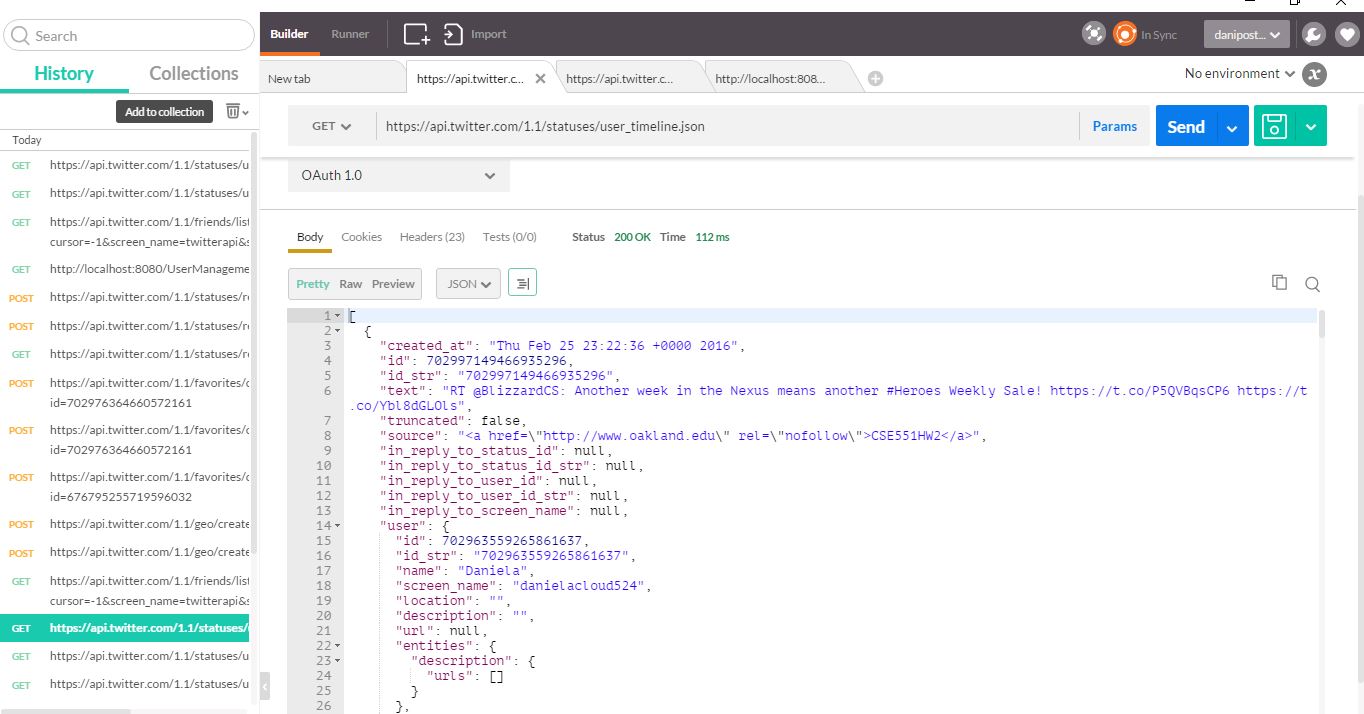
Destroy API:



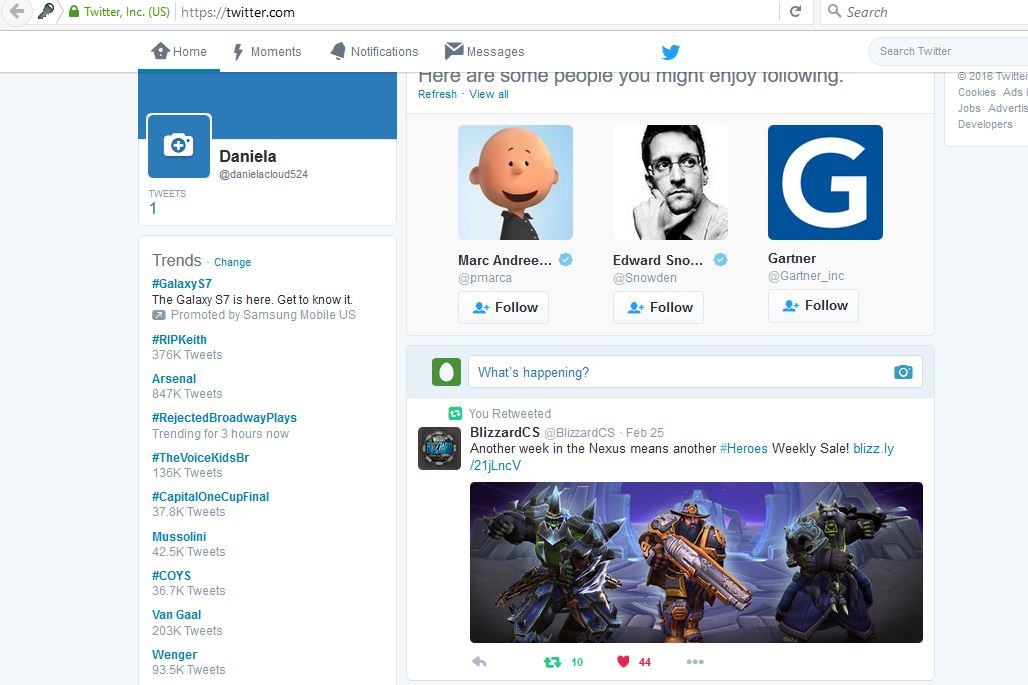
Update API:



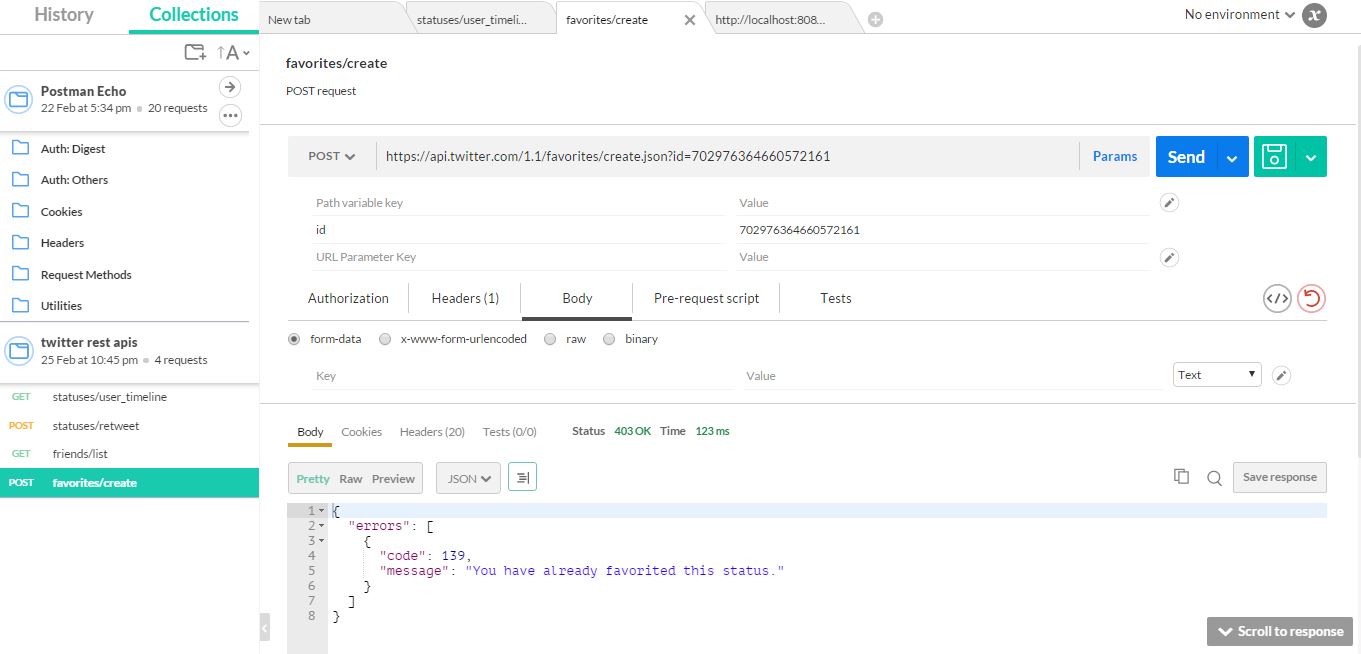
Get Status Timeline:



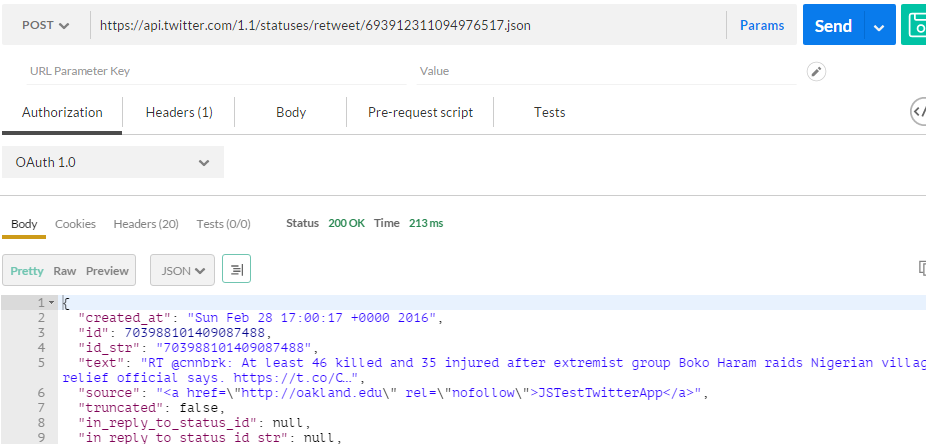
Favorite Create:



Favorites Create Error:



Status Retweet:





Friends List:

