Coding challenges for candidates

Here's some coding challenges we would like you to hack on the next few days. They are inspired by some of the real world problems we are working on a daily basis and should also give you a view on some of the potential task assignments you might be doing in the future.

We recommend using JavaScript and Node.js for the following challenge, because it is currently our tech of choice, but feel free to use some popular dynamic language you find the best suited for the case (e.g. Ruby, Python, Go which we can evaluate as well :))

Please return the assignments as a link to readable source code instead of sending files over email. You can use free source code hosting platforms like Github or Bitbucket. You can of course host the source files yourself.

1) Throttling API client

Write a throttled client library to be used with Facebook Graph API. You can include other helping packages, like Request but please write the actual throttling part yourself as it is the point of this exercise

Have a throttle limit of 600 requests per 600 seconds.
Send all the requests as they come (or wait if the throttle limit is reached)
Add the associated token to every request as a GET parameter token
Push received errors to a file 'request_log.txt'
Return promises for requests instead of traditional callbacks (You can use a library like Q for implementing this)

Offer an API that looks like:

```
var api = require('api');
api.endpoint = 'https://graph.facebook.com';
api.token = 'abcdfiruweferug';
```

```
api.get('/campaigns');

api.post('/campaigns', {
  'name': 'Some campaign',
  'budget': 500
});

api.put('/campaigns/123', {
  'budget': 1000
});

api.delete('/campaigns/123');
```

GET, POST, PUT & DELETE methods should return promise objects that implement then(onFulfilled, onRejected) method like this:

```
api.get('/campaigns').then(function(results) {
  // Handle results
}, function(error) {
  // Handle error
});
```

If a request fails, the promise will be rejected with an error message.

2) API endpoint for aggregation over SQL database

Your task is to create an API endpoint with the following request format:

```
GET /api/stats?ad_ids=1,2,3&start_time=2013-09-01&end_time=2013-10-01
```

The endpoint should take a number of `ad_ids` as GET parameters and report aggregate stats from the date range defined in parameters `start_time` & `end_time`. Results should be grouped by ad_id and have the actions reported as well.

The endpoint should respond with a JSON dict using the `ad_id` as a key.

Sample output:

```
{
  '1': {                    // ad_id as aggregation key
    'impressions': 412842,  // Sum of impressions
    'clicks': 21421,        // Sum of clicks
    'spent': 51234,         // Sum of spent
```

```
    'ctr': 0.0534,          // Click-through-rate
    'cpc': 80,              // Cost per click
    'cpm': 120,             // Cost per 1000 impressions
    'actions': {
      'mobile_app_install': {
        'count': 50,        // Sum of actions
        'value': 3900,      // Sum of action values
        'cpa': 520          // Cost per action
      },
      'page_like': {
        'count': 4,
        'value': 0,
        'cpa': 412
      }
    }
  },
  '2': {
    'impressions': 95345,
    'clicks': 924,
    'spent': 51242,
    'ctr': 0.0084,
    'cpc': 30,
    'cpm': 340,
    'actions': {}
  }
  ...
}
```

Implement a sample SQL database (we use MemSQL but you can go with MySQL) with a following schema:

Table: ad_statistics

Field	Type	Example value
ad_id	INT	1
date	DATE	2013-09-01
impressions	BIGINT	4123915
clicks	BIGINT	25190
spent	BIGINT	8291

Table: ad_actions

Field	Type	Example value
ad_id	INT	1

date	DATE	2013-09-01
action	VARCHAR	mobile_app_install
count	BIGINT	50
value	BIGINT	3900

You can fetch a data dump (in TSV format) for the statistics from here:

ad_statistics.tsv (52572 rows)
ad_actions.tsv (103602 rows)

For more info about how Facebook reports its statistics, you can check out:
https://developers.facebook.com/docs/reference/ads-api/adreportstats/

Bonus challenges to think about:

Design the code in a way that it supports concurrent queries and responds fast even if the tables would have millions of rows. Would you alter the data structures? How would you index the tables & fit as much as possible into memory? Any ideas on the data compression side?

How would you manage database connections for hundreds of concurrent API calls?