# RESTful JSON API

We include the following overview of our application programming interface in order to explain the calls and interfaces that we are employing in our program. This is aimed to give a comprehensive overview of all calls and explain their meanings.

**Authenticate**

**Method** - authToken

URL: /authenticate/authToken

Parameters: token

Request Type: GET

Format: {‘token’: “token unique ID”}

Response: { ‘Response’: ‘Success’}

Response: {‘Response’: ‘Fail’}

This call calls the server to verify the validity of a supplied token from the client. This will be used both in the situation of when a client enters a unique key to have a token assigned to their device, and the case when a device access a simulation and requires their token to be verified.

**Method** - authenticate/userLogin

Parameters: username, password

Request Type: GET

Format: {username: ‘ooo524’, password:’noneofyourbusiness’ }

Response: { Response: ‘Success’, client-auth-id: ‘kkj02’}

Response: { Response: ‘Fail’, client-auth-it: ‘’}

This call is to the server to verify the validity of a username and password pair from the client to the server. This is used in order to verify that the device accessing the application is an administrator.

**Add**

**Method** - AddDevice2Network

**URL** : /add/Device/Network

Request Type: POST

Parameters - : device\_name, network\_name, partition\_name, simulation\_name, token

Body Format:

{ token: ‘blahblah’, partition\_name: ‘Partition-1’, device\_name: 45, network\_name: 344, simulation\_name : ‘Sim02’, config\_map:{configmap format}}

This method informs the server that a user would like to add their device to a particular network in a particular partition in a particular simulation. This method requires the user’s token in order to verify validity of the device.

**Method** - Add2FreeList

**URL** : /add/Device/FreeList

Request Type: POST

Parameters - : device\_name, simulation\_name, token

Body Format:

{ token: ‘blahblah’, device\_name: 45, simulation\_name : ‘Sim02’}

This method informs the server that a user would like to add their device to the free list partition. This is a partition which has no networks and exists outside of all other networks and partitions in the simulation. This method requires the user’s token in order to verify validity of the device.

**Create**

**Method** - createNetwork

URL: /create/Network

Request Type: POST

Parameters: network\_name, simulation\_name, partition\_name, token

Body Format:

{ network\_name: ‘344’, simulation\_name: ‘Sim02’, partition\_name: ‘Parta’ token: ‘tokenID’}

This call informs the server that a new virtual network should be created within a partition. In the case that it is a device which is creating this virtual network, the token of that device is supplied in order to allow that device to manage that network.

**Method** - createDevice

URL: /create/Device

Request Type: POST

Parameters: device\_name, simulation\_name

Body Format:

{ device\_name: ‘devicew@mun.ca’, simulation\_name: ‘Sim02’}

This call informs the server that a new virtual device should be created within a partition.

**Method** - CreateSimulation

URL: /create/Simulation

Request Type: POST

Parameters: name, email\_list, num\_devices, num\_network, config\_map, tokenMethod

Body Format:

{ name: ‘Sim02’,

num\_devices: 11,

num\_networks: 5,

tokenMethod: ‘email’

config\_map: {

'Partition1':

{'networka' :

{ 'devicea' : 1, 'deviceb@mun.ca': 2, '[devicec@mun.ca](mailto:devicec@mun.ca)':3},

'networkb' :

{ 'deviced': 4, 'devicee': 5},

},

'Partition2':

{ 'networkc' :

{ 'devicef': 6, 'deviceg@mun.ca' : 7, 'deviceh@mun.ca': 8},

'networkd' :

{'devicei@mun.ca':9, 'device@mun.ca': 10},

},

'Partition3':

{ 'networke' : { 'devicek':'11'} }

freelist : {devicew: 13, devicex : 14}

}

This call informs the server that a new simulation is to be created. It includes the partitions, networks, and devices which should be part of this simulation. For each device, an email is entered, which will be used to propagate the token to.

**Method:** mergePartitions:

URL: /merge/Partitions

Parameters: partition\_a, partition\_b, simulation\_name

Reqest Type: POST

Format:

{ partition\_a: ‘parta’,

partition\_b : ‘partb’,

simulation\_name : ‘Sim01’,

}

This call informs the server that a new partition from two partitions is to be created. The partition is created by merging the two partitions together

**Delete**

**Method** - removeDevice

**URL** : /remove/Device/Network

Request Type: POST

Parameters - : device\_name, network\_name, partition\_name, simulation\_name

Body Format:

{ partition\_name: ‘Partition-1’, device\_name: 45, network\_name 344, simulation\_name : ‘Sim02’}

This method informs the server that a user would like to add their remove a device from a network to a particular network in a particular partition in a particular simulation. This method requires the user’s token in order to verify validity of the device.

**Method** - removeDevicefromFreeList

**URL** : /remove/Device/FreeList

Request Type: POST

Parameters - : device\_name, simulation\_name, token

Body Format:

{ token: ‘blahblah’, device\_name: 45, simulation\_name : ‘Sim02’}

This method informs the server that a user would like to remove their device to the free list partition. This is a partition which has no networks and exists outside of all other networks and partitions in the simulation. This method requires the user’s token in order to verify validity of the device.

**Method** - deleteDevice

URL: /delete/Device

Parameters: device\_name, simulation\_name

Request Type DELETE

Body Format:

{ device\_name: ‘45’, simulation\_name : ‘Sim02’ }

This call informs the server that a device should be deleted from the simulation.

**Method** - deleteNetwork

URL: /delete/Network

Parameters**:** network\_name, simulation\_id

Request Type: DELETE

Body Format:

{ networkname ‘344’, simulation\_name : ‘Sim02’ }

This call informs the server that a networks should be deleted from the simulation.

**Method** - deleteToken

URL: /delete/Token

Parameters: token

Request Type: DELETE

Body Format:

{token: ‘blahblah’ }

This call informs the server that a certain token should be removed from the database. All information with respect to this token is then wiped, including the device connected to it.

**Method:** deletePartition:

URL: /delete/Paritition

Parameters: partition\_name, simulation\_name, token

Reqest Type: DELETE

Format:

{ partition\_name: ‘parition02’,

simulation\_name : ‘Sim02’,

token: ‘blahblah’,

}

This call informs the server that it should delete a particular partition of the server. This will delete all networks within this partition as well.

**Method:** deleteSimulation

URL: /delete/Simulation

Parameters: simulation\_name

Reqest Type: DELETE

Format:

{ simulation\_name : ‘Sim02’,}

This call tells the server to delete an entire simulation. This deletes all data connected to this simulation including tokens, networks, partitions, and devices.

**Update**

**Method** - updateLocalCount

URL: /update/LocalCount

Parameters: localcounter, token, current\_network, simulation\_name

Request Type: POST

Body Format:

{token: ‘blahblah’, localcounter: 345, current\_network: ‘home’, ‘simulation\_name’ : ‘Sim02’ }

This method tells the server that the local count with respect to this device has been updated for this user with this specified token.

**Method** - updateNetworkName

URL: /update/NetworkName

Parameters: old\_name, new\_name, token, simulation\_name

Request Type: POST

Body Format:

{token: ‘blahblah’, old\_name: ‘oldie’, new\_name: ‘newbie’, simulation\_name: ‘Sim02’ }

This call tells the server to update a particular name of a network with a new supplied network name.

**Method:** updateDeviceName

URL: /update/DeviceName

Parameters: old\_name, new\_name, token, simulation\_name

Request Type: POST

Body Format:

{token: ‘blahblah’, old\_name: ‘oldie’, new\_name: ‘newbie’, simulation\_name: ‘Sim02’ }

This call tells the server to update the name of a particular device with the supplied name.

**Method**: updateSimulationName

URL: /update/SimulationName

Parameters: old\_name, new\_name, simulation\_name

Request Type: POST

Body Format:

{ old\_name: ‘oldie’, new\_name: ‘newbie’, simulation\_name: ‘Sim02’ }

This call tells the server to update this particular simulations name with the supplied name.

**Method**: updateTokenMethod

URL: /update/TokenMethod

Parameters: new\_method, simulation\_name

Request Type: POST

Body Format:

{ simulation\_name: ‘Sim02’, new\_method: ‘newbie’ }

This call tells the server to update the method by which tokens are sent to users with the supplied method.

**Method**: updateDeviceNumber:

URL: /update/DeviceNumber

Parameters: device\_number, simulation\_name

Request Type: POST

Body Format:

{ device\_number: 12, simulation\_name:’Sim02’ }

This call tells the server to update the number of devices in a simulation

**Method**: updateNetworkNumber

URL: /update/NetworkNumber

Parameters: network\_number, simulation\_name

Request Type: POST

Body Format:

{ network\_number: 12, simulation\_name: ‘Sim01’ }

This call tells the server to update the number of networks in a particular simulation.

**Method**: updateConfigMap:

URL: /update/ConfigMap

Parameters: partition\_name, config\_map, simulation\_name

Reqest Type: POST

Body Format:

{ PartitionName: ‘parition02’,

simulation\_name : ‘Sim01’,

,config\_map:

{ ‘networka’ :

{ ‘devicef’: 1, ‘deviceg’: 2, ‘deviceh’: 3},

‘networkd’ :

{‘devicei’: 4, ‘devicej’: 5},

},

}

This call tells the server to update a particular partition configuration within a particular simulation.

**Method:** divide/Partition:

URL: /divide/Partition

Parameters: partition\_name, network\_name , simulation\_name

Reqest Type: POST

Format:

{ partition\_name: ‘networka’,

simulation\_name : ‘Sim01’,

network: ‘networka’

}

This call informs the server that a new partition is to be created from a network within the partition

**GET**

**Method**: getSync

URL: /get/Sync

Parameters: eventQueue\* , token

Request Type: GET

Format:

{token : ‘blahblah’, simulation: ‘Sim02’ , eventQueue:

[

{ ‘URL’ : ‘/add/Device’, ‘Body’ : format\_body, timestamp : ‘2015-01-09 12:00:00’},

{ ‘URL’ : ‘/add/Device’ , ‘Body’ : format\_body, timestamp : ‘2015-01-09 12:03:00’},

{‘URL’ : ‘/delete/Device’ , ‘Body’ : format\_body, timestamp : ‘2015-01-09 12:23:00’},

{‘URL’ : ‘ /add/Simulation’ , ‘Body’ : format\_body, timestamp : ‘2015-01-09 12:33:00’},

{‘URL’ : ‘/update/LocalCount’, ‘Body’ : format\_body, timestamp : ‘2015-01-012:44:00’}

]

}

This method is the core of our system. It sends the status on the client side, including all events which have occurred on the client side since the last getSync to the server. This information is then processed by the server and the server is updated. Once this is done, the current global status of the simulation is returned to the device so update the device.

Method: getSimulation

URL: /get/Simulation

Parameters: simulation\_name

Request Type: GET

Format:

{client\_id : ‘blahirinr902enrg’, simulation\_name : ‘Sim02’}

Method: getStates

URL: /get/States

Parameters: client\_id, simulation\_name

Request Type: GET

Format:

{client\_id : ‘blahirinr902enrg’, simulation\_name : ‘Sim02’}

**\*see above for the format body of the event queues**

**RESPONSE** : The Application state is returned as a JSON object for all calls

**links**: <http://crunchify.com/how-to-iterate-through-jsonarray-in-javascript/>

<http://code.flickr.net/2008/09/26/flickr-engineers-do-it-offline/>

<http://code.flickr.net/2009/03/18/building-fast-client-side-searches/>

**Example 1: A typical user. The token is used to compare the values in the database and gives the user a state based on this and the system and simulation state. Don’t worry about the clientid field, as no one will have one until we use start getting access rights such as administrators, super admins, etc.**

{ appstate : {

device : {

token: ‘blahblah’

email: ‘[ooo524@mun.ca](mailto:ooo524@mun.ca)’,

verified : true,

current\_partition: ‘Partition\_1’,

current\_network : ‘networka’,

current\_simulation: ‘Sim02’,

registeredOn : ‘2015-01-09’,

is\_admin : false,

networks\_created: [‘networke’],

application\_id: ‘default’,

localcount: 23,

globalcount: 345,

current\_device\_name: ‘device11’,

activity: ‘’,

},

current\_simulation\_session : {

num\_devices: 11,

id : ‘blahdu3’

num\_networks: 5,

simulation\_name: ‘Sim02’,

config\_map: {

'Partition1':

{'networka' :

{ 'devicea' : 1, 'deviceb@mun.ca': 2, '[devicec@mun.ca](mailto:devicec@mun.ca)':3},

'networkb' :

{ 'deviced': 4, 'devicee': 5},

},

'Partition2':

{ 'networkc' :

{ 'devicef': 6, 'deviceg@mun.ca' : 7, 'deviceh@mun.ca': 8},

'networkd' :

{'devicei@mun.ca':9, 'device@mun.ca': 10},

},

'Partition3':

{ 'networke' : { 'devicek':'11'}

}

freelist : { 'devicef': 11, 'devicen@mun.ca' : 12, 'deviceo@mun.ca': 13 }

},

globalcount: 230,

simulation\_population: 3,

token\_list {},

activity\_logs: ‘blahblah some stuff happened in this simulation, data only about your actions ‘

},

application: {

simulation\_list: [{name : ‘Sim01’ , num\_devices: 11, num\_networks: 5},

{name: ‘‘Sim02’, num\_devices: 45, num\_networks: 4},

{name: ‘Sim03’, num\_devices: 9,num\_networks: 2},

{name: ‘Sim04’, num\_devices: 7,num\_networks: 2},

{name: ‘Sim05’ ,num\_devices: 2, num\_networks: 8}]

super\_admin: {},

total\_devices : 68,

total\_networks: 23,

},

‘states’ :

{id : ‘blahdu3’,

state: [{ ‘id’ : simulation\_session, timestamp: ‘2015-01-012:44:00’

‘devices’: [{ device: device\_object},{ device: device\_object},{ device: device\_object},{ device: device\_object}],

{‘id’ : simulation\_session, timestamp: ‘2015-02-013:44:00’,

‘devices’: [{ device: device\_object},{ device: device\_object},{ device: device\_object},{ device: device\_object}],

{ ‘id’ : simulation\_session, timestamp: ‘2015-02-022:44:00’ ,

‘devices’: [{ device: device\_object},{ device: device\_object},{ device: device\_object},{ device: device\_object}],

} ]

}

}

**Example 1-a: The same as before but the user is only registered in one simulation. The token is used to compare the values in the database and gives the user a state based on this and the system and simulation state.**

{ appstate : {

device : {

tokens: ‘blahblah’,

email: ‘[ooo524@mun.ca](mailto:ooo524@mun.ca)’,

verified : true,

currentpartition: ‘Partition\_1’,

currentnetwork : ‘networka’,

networks\_created: [],

currentsimulation: ‘Sim02’,

registeredOn : ‘2015-01-09’,

is\_admin : false,

application\_id: ‘default’,

current\_device\_name: ‘device11’,

activity: ‘’,

},

current\_simulation\_session : {

num\_devices: 11,

id : ‘blahdu3’,

num\_networks: 5,

simulation\_name: ‘Sim02’,

config\_map: {

{‘Partition\_1’,

{ ‘networka’ :

{ ‘devicea@mun.ca’ : ‘1’, ‘deviceb@mun.ca’: 2, ‘devicec@mun.ca’:3},

‘networkb’ :

{ ‘deviced@mun.ca’: 4, ‘devicee@mun.ca’: 5},

},

Partition\_2:

{ ‘networkc’ :

{ ‘devicef@mun.ca’: 6, ‘deviceg@mun.ca’ : 7, ‘deviceh@mun.ca’: 8},

‘networkd’ :

{‘devicei@mun.ca’: 9, ‘devicej@mun.ca’: 10},

},

Partition\_3:

{ ‘networke’ :

{ ‘devicek@mun.ca’:11}

}

freelist : { 'devicef': 11, 'devicen@mun.ca' : 12, 'deviceo@mun.ca': 13 }

]},

globalcount: 230,

simulation\_population: 3,

token\_list {},

activity\_logs: ‘blahblah some stuff happened in this simulation, data only about your actions ‘

},

application: {

simulation\_list: [{name : ‘Sim01’ , num\_devices: 11, num\_networks: 5},

{name: ‘‘Sim02’, num\_devices: 45, num\_networks: 4},

{name: ‘Sim03’, num\_devices: 9,num\_networks: 2},

{name: ‘Sim04’, num\_devices: 7,num\_networks: 2},

{name: ‘Sim05’ ,num\_devices: 2, num\_networks: 8}]

super\_admin: {},

total\_devices : 68,

total\_networks: 23,

},

‘states’ :

{id : ‘blahdu3’,

` state: [{ ‘id’ : simulation\_session, timestamp: ‘2015-01-012:44:00’

‘devices’: [{ device: device\_object},{ device: device\_object},{ device: device\_object},{ device: device\_object}]

{‘id’ : simulation\_session, timestamp: ‘2015-02-013:44:00’,

‘devices’: [{ device: device\_object},{ device: device\_object},{ device: device\_object},{ device: device\_object}]

{ ‘id’ : simulation\_session, timestamp: ‘2015-02-022:44:00’ ,

‘devices’: [{ device: device\_object},{ device: device\_object},{ device: device\_object},{ device: device\_object}]

} ]

}

}

**Example 2: A new user. The Token is used to compare the values in the database and gives the user a state based on this and the system and simulation state.**

{ appstate : {

device : {

tokens: ‘’,

email: ‘’

verified : false,

currentnetwork : ‘’,

currentpartition: ‘’,

currentsimulation: ‘’,

registeredOn : ‘2015\_01-09’,

networks\_created: [],

is\_admin : false,

application\_id: ‘default’,

current\_device\_name: ‘’,

activity: ‘’

},

current\_simulation\_session : {

num\_devices: 0,

num\_networks: 0,

id : ‘blahdu3’,

simulation\_name: ‘’ ,

config\_map: {},

globalcount: 0,

simulation\_population: 0,

token\_list {},

activity\_logs: ‘’

},

application: {

simulation\_list: [{name : ‘Sim01’ , num\_devices: 11, num\_networks: 5},

{name: ‘‘Sim02’, num\_devices: 45, num\_networks: 4},

{name: ‘Sim03’, num\_devices: 9,num\_networks: 2},

{name: ‘Sim04’, num\_devices: 7,num\_networks: 2},

{name: ‘Sim05’ ,num\_devices: 2, num\_networks: 8}]

super\_admin: {},

total\_devices : 68,

total\_networks: 23,

},

‘states’ :

{id : ‘blahdu3’,

` state: [{ ‘simulation’ : simulation\_session, timestamp: ‘2015-01-012:44:00’

‘devices’: [{ device: device\_object},{ device: device\_object},{ device: device\_object},{ device: device\_object}]

{‘simulation’ : simulation\_session, timestamp: ‘2015-02-013:44:00’,

‘devices’: [{ device: device\_object},{ device: device\_object},{ device: device\_object},{ device: device\_object}]

{ simulation’ : simulation\_session, timestamp: ‘2015-02-022:44:00’ ,

‘devices’: [{ device: device\_object},{ device: device\_object},{ device: device\_object},{ device: device\_object}]

} ]

}

}

**Example 3: A super user. The user’s password and username is used to compare the values in the database and gives the user a state based on this and the system and simulation state. The username and password is swapped for a client side clientid to identify the user.**

**\*\*\* Should not be implemented as of yet \*\*\*\***

{ appstate : {

device : {

tokens: ‘’,

email: ‘’

verified : true,

currentnetwork : ‘’,

simulation: ‘’,

currentpartition: ‘’,

networks\_created: [ ],

registeredOn : ‘2015-01-09’

**is\_admin : true,**

application\_id: ‘default’

current\_device\_name: ‘’,

activity: ‘’,

},

current\_simulation\_session : {

num\_devices: ,

num\_networks: ,

simulation\_name: ‘’,

id : ‘blahdu3’,

config\_map: {},

globalcount: 0,

simulation\_population: 0,

token\_list { },

},

application: {

simulation\_list: [{name : ‘Sim01’ , num\_devices: 11, num\_networks: 5},

{name: ‘‘Sim02’, num\_devices: 45, num\_networks: 4},

{name: ‘Sim03’, num\_devices: 9,num\_networks: 2},

{name: ‘Sim04’, num\_devices: 7,num\_networks: 2},

{name: ‘Sim05’ ,{num\_devices: 2, num\_networks: 8}]

super\_admin: {},

total\_devices : 68,

total\_networks: 23

},

‘state’ :

{id : ‘blahdu3’,

` states: [{ ‘’ : simulation\_session, timestamp: ‘2015-01-012:44:00’

‘devices’: [{ device: device\_object},{ device: device\_object},{ device: device\_object},{ device: device\_object}]

{‘id’ : simulation\_session, timestamp: ‘2015-02-013:44:00’,

‘devices’: [{ device: device\_object},{ device: device\_object},{ device: device\_object},{ device: device\_object}]

{ ‘id’ : simulation\_session, timestamp: ‘2015-02-022:44:00’ ,

‘devices’: [{ device: device\_object},{ device: device\_object},{ device: device\_object},{ device: device\_object}]

} ]

}

**Explanations**

Right now when the application starts fresh, the user navigates to the webpage for the first time and the client makes a call at start. This call is to getSync with the server. At this stage the client will not have a token. The will check the token ID sent. If it is empty it sends back this state every time. Later on when we implement client ID, this may change.

{ appstate : {

device : {

tokens: ‘’,

email: ‘’

verified : false,

currentnetwork : ‘’,

currentpartition: ‘’

currentsimulation: ‘’,

registeredOn : ‘2015-01-09’

networks\_created: [ ]

is\_admin : false,

application\_id: ‘default’

current\_device\_name: ‘’,

},

current\_simulation\_session : {

num\_devices: 0,

num\_networks: 0,

simulation\_name: ‘’ ,

id : ‘blahdu3’,

config\_map: {

map: { }

},

globalcount: 0,

simulation\_population: 0,

token\_list {},

activity\_logs: ‘’

},

application: {

simulation\_list: [{ name : ‘Sim01’ ,

num\_devices: 11,

num\_networks: 5}

]

num\_devices : 68,

num\_networks: 23,

superadmin: {} }

}

}

**Create Simulations**

If there are no Simulations in the environment that simulation\_list at the end will also be blank.

Now after the user sends a call to create a simulation. The follow occurs

**Step 1:**

Initiator: Client

Client sent the request with an event queue. Embedded in that queue is the the URL for /create/Simulation.

**Step 2:**

Initiator: Server

Server routes the clients request for getSync and parses the token and the eventqueue. Once the server checks the URL contained in the eventQueue it extracts the information as dictated in the above API for how the format\_body will look like. The server extracts the format body. This will be of the form:

{ name: ‘Sim02’,

num\_devices: 11,

num\_networks: 5,

id : ‘blahdu3’,

tokenMethod: ‘email’

config\_map: { Partition\_1:

{ ‘networka’ :

{ ‘devicea@mun.ca’ : 1, ‘deviceb@mun.ca’: 2, ‘devicec@mun.ca’:3},

‘networkb’ :

{ ‘deviced@mun.ca’: 4, ‘devicee@mun.ca’: 5},

},

Partition\_2:

{ ‘networkc’ :

{ ‘devicef@mun.ca’: 6, ‘deviceg@mun.ca’ : 7, ‘deviceh@mun.ca’: 8},

‘networkd’ :

{‘devicei@mun.ca’: 9, ‘device@mun.caj’: 10},

},

Partition\_3:

{ ‘networke’ :

{ ‘devicek@mun.ca’:11}

}

}

The server extracts the necessary fields of email\_list, config\_map, tokenMethod, num\_networks, num\_devices, name.

The server then calls the appropriate method indicated by the URL. In this case it is

createSimulation( email\_list, config\_map, tokenMethod, num\_networks, num\_devices, name).

**Step 3:**

Initiator: Simulation Manager

Once the Simulation Manager receives the call for createSimulation(params) it needs to do several things:

First, connect to the database and enter the information provided as an object. However it needs to put it back in the database in the format of:

{ name: ‘Sim02’,

num\_devices: 11,

num\_networks: 5,

tokenMethod: ‘email’

config\_map: { Partition\_1:

{ ‘networka’ :

{ ‘devicea@mun.ca’ : ‘1’, ‘deviceb@mun.ca’: 2, ‘devicec@mun.ca’:3},

‘networkb’ :

{ ‘deviced@mun.ca’: 4, ‘devicee@mun.ca’: 5},

},

Partition\_2:

{ ‘networkc’ :

{ ‘devicef@mun.ca’: 6, ‘deviceg@mun.ca’ : 7, ‘deviceh@mun.ca’: 8},

‘networkd’ :

{‘devicei@mun.ca’: 9, ‘devicej@mun.ca’: 10},

},

Partition\_3:

{ ‘networke’ :

{ ‘devicek@mun.ca’:11}

}

free\_partition:

}

This is inserted in the collection for simulations. Once this is done, the manager gets the devices in the simulation using an iterator. This iterator will be shared code on the client and server.

For each device it needs to create a new object in the format:

user : {

token: ‘’,

email: ‘’

clientid: ‘’, \*\* don’t worry about how or when we will implement this \*\*

verified : false,

currentnetwork : ‘’,

currentpartition: ‘’

currentsimulation: ‘’,

registeredOn : ‘’

networks\_created: {}

is\_admin : false,

application\_id: ‘default’

current\_device\_name: ‘’,

},

To create this object this is the same as:

*var user\_data = {}*

As shown above verified is obviously false because the user has not registered with the simulation yet. But the device technically exists now in the virtual simulation.

It also needs to modify the following fields in this Object.

name is the name of the simulation provided by the parameters

To set the currentSimulation this user/device has been created in.

*user.currentsimulation = name;*

To set the currentNetwork \_ or default network at start - this user/device has been created in. The client has a function called getparent(node) to access the network name from a given device name. the device name is passed in as the node. This is because config\_map is in a tree structure.

*user.currentnetwork = getparent(device\_name);*

We set the current device name as

*user.current\_device\_name = device\_name*

we set the current partition for the device as

*user.currentpartition = getparent(user.currentnetwork)*

We set the email of the device to the name of the device as well

*user.email = device\_name*

And Finally most importantly we assign the user the token we just generated. We don’t need to actually pass in the simulation name but just as an example. Token generation is a blackbox.

user.token = tokenManager.generateToken(name)

**Step 4:**

Initiator: Database Manager

Thats it now we ship this to the database.

First wrap it up as a JSON object.

so

for\_db\_user = JSON.parse(user);

mongo.user.insert(for\_db\_user);

Do this for each user to be created

**Step 5:**

Initiator: Simulation Manager

The application must now edited to include the new simulation. A new object must be entered in the array of simulations at this level.

item = { ‘name’ : *simulation.name*, ‘num\_devices’ : *simulation.num\_devices*, ‘num\_networks’ : *simulation.num\_networks* }

**Step 6:**

Initiator: Database Manager

Retrieve the application collection object from the database. This object looks like:

application: {

simulation\_list: [{name : ‘Sim01’ , num\_devices: 11, num\_networks: 5},

{name: ‘‘Sim02’, num\_devices: 45, num\_networks: 4},

{name: ‘Sim03’, num\_devices: 9,num\_networks: 2},

{name: ‘Sim04’, num\_devices: 7,num\_networks: 2},

{name: ‘Sim05’ ,{num\_devices: 2, num\_networks: 8}]

super\_admin: {},

total\_devices : 68,

total\_networks: 23,

}

Then send back the response to the simulation manager.

**Step 7:**

Initiator: Simulation Manager

Add the item from above in the simulation\_list array.

simulation\_list.push(item)

edit the total\_devices and add the simulation.num\_devices to it

application.total\_devices += simulation.num\_devices

Do the same for the num\_networks

application.total\_networks += simulation.num\_networks

**Check above for response format.**

**Auth Token**

Step 1:

Initiator: Client

Client sends the request for authentication called /authenicate/Token with its necessary body: the token.

Step 2:

Initiator: Server

Server parses request, extracts the token from the JSON object and calls the function Simulation.Manager.authenticate(token). The actual function name is arbitrary. What matters is what the function call DOES.

Step 3:

Initiator: Simulation Manager

Simulation manager asks the database manager for the item in the ‘users’ collection that has the token matching the one it has.

If the database returns an empty object then the server must return false, otherwise the server returns true.

Step 4:

Initiator: Server

The server sends this response back as a JSON object

**Steps for activity Logs**

The application should be able to display the activity logs of the simulation not only within the device view but an overview for the administrator of the simulation.

To implement this we have included the activity parameter for the activity in each device collection to display activities performed by that device. The activity log is also kept for each simulation in the current\_simulation collection.