1. Install Neo4J on the server:  
     
   46.101.180.63:7474/browser/  
     
   Run with command (from ‘/’):   
     
   neo4j console  
     
   - will run the ‘neo4j service’ on the console, and we stop it by pressing “crtl+c”  
     
   neo4j start  
     
   - it will run, but- in the background. To stop it use command “Neo4J stop”   
     
   TMUX COMMANDS: <https://gist.github.com/henrik/1967800>  
     
   tmux detach

1. To create the knowledge graph:

* Unable to create the graph without actual data

**Steps from here:**

1. **~~Make fake data on Neo4J for presentation purposes (Done)~~**
2. **~~Create a Cypher file, that takes data from a source, maybe from csv- or from postgres- or from the wrappers~~**
3. **~~Get all users Details from One name(Done)~~**
4. **~~Add user to Neo4j(Done)~~**
5. **~~get BTC address details of each user(BlockExplorer API)(Done)~~**
6. **~~Add BTC address to Neo4j(Done)~~**
7. **~~Relate user with BTC Adress(Done)~~**
8. **~~Get all the transactions that BTC has(Done)~~**
9. **~~Add transactions to neo4j(Done)~~**
10. **~~Get all the input Address in those transactions (Done)~~**
11. **~~And add addresses to Neo4j(Done)~~**
12. **~~Create Inputs relation between addresses and transaction(Done)~~**
13. **~~Get Output addresses in a transaction(Done)~~**
14. **~~And add addresses to Neo4j(Done)~~**
15. **~~Create Outputs relation between addresses and transaction(Done)~~**

**NOTE: we are getting two level down using the addresses form one name**

**LVL1: one name**

1. **~~This will take the data and create a populated knowledge graph(Done)~~**
2. **~~Get twitter followers and add them to the graph (Mathew)(Done)~~**
   1. **~~Get twitter and add 2 graph (Done)~~**
   2. **~~Add Relation between person and twitter(similar to person and BTC)(Done)~~**
   3. ***~~Add relation between the twitter of a person and his followers~~* ~~(Done)~~**
3. Add Blondie Ontology to Neo4j (Firas) time to solve till Wed(21/09/2016)
4. **~~Add Blocks node to Graph (Firas) (Matthew) Done~~**
5. Apply the Blocks change
6. Adding Time Tree to Neo4j (**Optional**) (Firas)
7. After creating the knowledge graph- we will create also a Cypher file for an endpoint (**OPTIONAL**)**(Search for online projects)**
8. **~~Create a Cypher file that will take the data from Neo4J and put it in our website (Firas) (OPTIONAL)(Its not applicable : Sol 11)~~**
9. **~~//Create new script to implement visualization(done)~~**
10. Also look for queries using Cypher: what type of queries would be the same ones as Ernane talked about?
11. Export to d3.js visualization with Statistics **Postponed until end of (6)**
12. Create on the fly wrapper that adds Btc transaction to the Graph
    1. **~~Import data(Done)~~**
    2. Add data to graph
13. Adding Abe instead of toshi and using it in D2RQ (Doing this will be finishing the work ernane) and then we add it to the website
14. **I~~MPORTANT: Creating website that contains every thing we have done(Details check~~** [**~~Semantic blockchain website~~**](https://docs.google.com/document/d/1rR4BSNh05sOnAGgtCh_aZdCkJmZhwCcPqz154BiVigs/edit)**~~).(Monday- To be done by Ernane so he says)~~**
15. **Parameters: (**
16. *Main statistics*
    1. *Size of data*
    2. *Number of users*
    3. *NUmber of transactions*
    4. *Number of transactions per user (histogram)*
17. *Obtain the set of edges defined with transactions between users*
    1. *UserA, UserB (#number of transactions) (#time of transactions)*

LINKS:   
  
<https://neo4j.com/blog/cypher-load-json-from-url/>

<http://stackoverflow.com/questions/38361325/import-json-to-neo4j-using-py2neo>

**Onename API INFO:**   
  
*App ID*:   
6a3f8ffc7bd83d68723c02e26b486981

*Your App Secret*:   
a98b34613a258c6610f3b143c788a2e37a8878cfb444a8ea9eacc3af4b073700

QUERIES:   
start n=node(\*)

match (n:BTC\_Address)

return count(n)

match (n:BTC\_Address)

match (p:BTC\_Address)

match (t:Transaction)

match (n)-[:Inputs\_to]->(t)

match (t)-[:Outputs\_to]->(p)

return t,p,n

limit 25

Create(p:Person {name:"Firas Kassawat",born:1989,PostalAddress:"Nordrhein-Westfalen",description:"The universe is change; our life is what our thoughts make it"})

return p

Create(a:BTC\_Address {address:"1N385y2UaXVPKsL7udGX667k6Xkbp1K3Lc",hash\_160:"e6c1635e7d89402183a83bc33cb81cd79eb01ac6",Total\_Received:"235.19",total\_sent:"235.19"})

return a

Create(t:Twitter {handle:"firaskassawat",name:"Firas Kassawat", url:"http://t.co/HOwTHZ0Pb6", location:"Bonn", followers: 2792, following: 890, profile\_image\_url: "http://pbs.twimg.com/profile\_images/752597956985888769/Phgyfbmf\_normal.jpg"})

return t

Create(a:Transaction

{hsh:"202426439a5d82fa3779c3d0417c3a300670934833a083c460a727d6f38a2255",Size:"1912",Received\_Time:"2016-08-30 23:35:09",total\_in:"2,266,471.73 ",total\_out:"2,265,836.19 ",fee:"635.54"})

return a

Create(a:Input\_Transaction

{hsh:"202426439a5d82fa3779c3d0417c3a300670934833a083c460a727d6f38a2255",Size:"1912",amount:"2,266,471.73",id:1126 ,script:" 483045022100961431763509CEEC41562041165E8AC187A9CC05EF949474A53AD791FD264B8202207D60BD399432601ECE848509758906968E1959B7DF50CE3D1BFBD93C5B829CDB012103286EEFB21E4E287600A7819EF3C16AD36A76A7417F84EEC823C9A5A446FD4F22"})

return a

Create(a:Output\_Transaction

{hsh:"202426439a5d82fa3779c3d0417c3a300670934833a083c460a727d6f38a2255",Size:"1912",amount:"2,265,836.19",id:1126 ,script:" 483045022100961431763509CEEC41562041165E8AC187A9CC05EF949474A53AD791FD264B8202207D60BD399432601ECE848509758906968E1959B7DF50CE3D1BFBD93C5B829CDB012103286EEFB21E4E287600A7819EF3C16AD36A76A7417F84EEC823C9A5A446FD4F22"})

return a

**Note: for a certain transaction the output and input have the same id and same hash**

MATCH (p:Person {name:'Firas Kassawat'}), (b:BTC\_Address {address:'1N385y2UaXVPKsL7udGX667k6Xkbp1K3Lc'})

CREATE (p)-[:HAS]->(b)

Delete a relationship:

START n=node(\*)

MATCH (n)-[rel:Follows]->(r)

DELETE rel

// query for a transaction with btc and person if exist

MATCH (n:BTC\_Address)-[r:Inputs\_to]->(m:Transaction)

MATCH (m:Transaction)-[x:Outputs\_to]->(p:BTC\_Address)

where p<>n AND m.hsh="2e9f40d4af66964a526d51233f87c401eaf66e05afd1e65928d2f2a51e636232"

Optional MATCH (a:Person)-[h:HAS]->(p)

Optional MATCH (b:Person)-[h:HAS]->(n)

RETURN n,r,m,x,p,a,b

LIMIT 50;

import os

import requests

import py2neo

# set up authentication parameters

py2neo.authenticate("46.101.180.63:7474", "neo4j", "uni-bonn")

# Connect to graph and add constraints.

neo4jUrl = os.environ.get('NEO4J\_URL',"http://46.101.180.63:7474/db/data/")

graph = py2neo.Graph(neo4jUrl)

# Add uniqueness constraints.

graph.run("CREATE CONSTRAINT ON (q:Person) ASSERT q.id IS UNIQUE;")

# Build URL.

apiUrl = "https://api.onename.com/v1/users/fredwilson?app-id=demo-app-id&app-secret=demo-app-secret"

# apiUrl = "https://raw.githubusercontent.com/s-matthew-english/26.04/master/test.json"

# Send GET request.

json = requests.get(apiUrl, headers = {"accept":"application/json"}).json()

# print(json)

# Build query.

query = """

RETURN {json}

"""

# Send Cypher query.

# py2neo.CypherQuery(graph, query).run(json=json)

# graph.run(query).run(json=json)

graph.run(query,json=json)