In [1]:

```
import networkx as nx
import matplotlib.pyplot as plt
import pandas as pd
from operator import itemgetter
import seaborn as sns
import numpy as np
from networkx.algorithms import node_classification
from networkx.algorithms import community
```

In [2]:

```
import warnings
warnings.filterwarnings('ignore')
```

In [3]:

```
# Reading datasets and importing data
G = nx.Graph(day="Stackoverflow")
df_nodes = pd.read_csv("stack_network_nodes.csv")
df_edges = pd.read_csv("stack_network_links.csv")
for index, row in df_nodes.iterrows():
    G.add_node(row['name'], group=row['group'], nodesize=row['nodesize'])

for index, row in df_edges.iterrows():
    G.add_weighted_edges_from([(row['source'], row['target'], row['value'])])
```

In [4]:

```
#Nodes dataframe
df_nodes.head(10)
```

Out[4]:

	name	group	nodesize
0	html	6	272.45
1	css	6	341.17
2	hibernate	8	29.83
3	spring	8	52.84
4	ruby	3	70.14
5	ruby-on-rails	3	55.31
6	ios	4	87.46
7	swift	4	63.62
8	html5	6	140.18
9	c	1	189.83

In [5]:

```
# edges dataframe
df_edges.head(10)
```

Out[5]:

	source	target	value
0	azure	.net	20.933192
1	sql-server	.net	32.322524
2	asp.net	.net	48.407030
3	entity-framework	.net	24.370903
4	wpf	.net	32.350925
5	linq	.net	20.501744
6	wcf	.net	28.074400
7	c#	.net	62.167895
8	tdd	agile	37.146590
9	codeigniter	ajax	23.191900

In [6]:

```
#general info of network
print(nx.info(G))
```

```
Name:
Type: Graph
Number of nodes: 115
Number of edges: 245
Average degree:   4.2609
```

In [7]:

```
print("\nList of all {} nodes present\n{}".format(len(G.nodes()), G.nodes()))
```

```
List of all 115 nodes present
['html', 'css', 'hibernate', 'spring', 'ruby', 'ruby-on-rails', 'ios', 'swif
t', 'html5', 'c', 'c++', 'asp.net', 'c#', 'objective-c', 'javascript', 'jque
ry', 'redux', 'reactjs', 'php', 'mysql', 'spring-mvc', '.net', 'react-nativ
e', 'spring-boot', 'less', 'sass', 'hadoop', 'apache-spark', 'sql-server',
'express', 'node.js', 'mongodb', 'iphone', 'github', 'git', 'excel', 'excel-
vba', 'entity-framework', 'linq', 'wcf', 'wpf', 'android', 'java', 'scala',
'ajax', 'django', 'python', 'vba', 'xcode', 'apache', 'nginx', 'angularjs',
'asp.net-web-api', 'laravel', 'plsql', 'oracle', 'json', 'xml', 'flask', 'wo
rdpress', 'java-ee', 'maven', 'jsp', 'bash', 'linux', 'angular2', 'typescrip
t', 'codeigniter', 'tdd', 'agile', 'twitter-bootstrap', 'web-services', 'res
t', 'testing', 'selenium', 'android-studio', 'redis', 'jenkins', 'docker',
'amazon-web-services', 'angular', 'osx', 'machine-learning', 'qt', 'window
s', 'ubuntu', 'ionic-framework', 'elasticsearch', 'vue.js', 'r', 'embedded',
'go', 'visual-studio', 'postgresql', 'sql', 'unix', 'eclipse', 'vb.net', 'un
ity3d', 'devops', 'drupal', 'shell', 'bootstrap', 'xamarin', 'azure', 'mvc',
'haskell', 'api', 'twitter-bootstrap-3', 'regex', 'perl', 'cloud', 'photosho
p', 'powershell', 'matlab']
```

In [8]:

```
# check whether our network is connected or not connected
if nx.is_connected(G):
    print('Connected Graph')
else:
    print("Not connected")
```

```
Not connected
```

In [9]:

```
# Network Density
print("\nNetwork density:", nx.density(G))
# network density of 0.037376 says that it is not completely connected.
# May not all the nodes are connected
```

```
Network density: 0.03737604881769641
```

In [10]:

```
# diameter of largest component
components = nx.connected_components(G)
largest_component = max(components, key=len)
print(largest_component)
subgraph = G.subgraph(largest_component)
diameter = nx.diameter(subgraph)
print("\nNetwork diameter of largest component:", diameter)
```

```
{'elasticsearch', 'postgresql', 'visual-studio', 'typescript', 'laravel', 't
witter-bootstrap-3', 'embedded', 'php', 'vb.net', 'reactjs', 'swift', 'sprin
g-boot', 'amazon-web-services', 'azure', 'ajax', 'apache', 'less', 'python',
'spring-mvc', 'wpf', 'machine-learning', 'flask', 'java', 'mvc', 'redux', 'd
jango', 'jenkins', 'twitter-bootstrap', 'nginx', 'sass', 'objective-c', 'ang
ular2', 'unix', 'spring', 'vue.js', 'git', 'r', 'bootstrap', 'android', 'web
-services', 'entity-framework', 'maven', 'devops', 'bash', 'asp.net-web-ap
i', 'eclipse', 'asp.net', 'xcode', 'windows', 'ios', 'redis', 'ionic-framewo
rk', 'unity3d', 'matlab', 'linux', 'plsql', 'javascript', 'node.js', 'mongod
b', '.net', 'iphone', 'docker', 'c++', 'powershell', 'android-studio', 'cs
s', 'wordpress', 'shell', 'angularjs', 'mysql', 'linq', 'xamarin', 'go', 'sq
l', 'xml', 'codeigniter', 'oracle', 'angular', 'qt', 'html', 'java-ee', 'res
t', 'json', 'osx', 'drupal', 'html5', 'c', 'ruby-on-rails', 'photoshop', 'cl
oud', 'c#', 'ruby', 'jquery', 'react-native', 'github', 'jsp', 'wcf', 'ubunt
u', 'api', 'express', 'hibernate', 'sql-server'}

Network diameter of largest component: 10
```

In [11]:

```
# Triadic closure(Local Clustering coefficient). This is one of the link prediction methods
triadic_closure = nx.transitivity(G)
print("\nTriadic closure:", triadic_closure)
```

```
Triadic closure: 0.48709239130434784
```

Centrality

In [12]:

```
# Degree Centrality
# Top three nodes having the highest Degree Centrality
def find_nodes_with_highest_deg_cent(G):
    # Compute the degree centrality of G: deg_cent
    deg_cent = nx.degree_centrality(G)
    # Compute the maximum degree centrality: max_dc
    max_1_dc = max(list(deg_cent.values()))
    max_2_dc = list(sorted(deg_cent.values()))[-2]
    max_3_dc = list(sorted(deg_cent.values()))[-3]

    maxnode1 = set()
    maxnode2 = set()
    maxnode3 = set()

    # Iterate over the degree centrality dictionary
    for k, v in deg_cent.items():

        # Check if the current value has the maximum degree centrality
        if v == max_1_dc:
            # Add the current node to the set of nodes
            maxnode1.add(k)
        if v == max_2_dc:
            # Add the current node to the set of nodes
            maxnode2.add(k)
        if v == max_3_dc:
            # Add the current node to the set of nodes
            maxnode3.add(k)

    return maxnode1, maxnode2, maxnode3
top_deg_dc, top2_deg_dc, top3_deg_dc = find_nodes_with_highest_deg_cent(G)
print("\nTop three nodes having the highest degree centrality :", top_deg_dc, top2_deg_dc,
      top3_deg_dc)
```

```
Top three nodes having the highest degree centrality : {'jquery'} {'css', 'c
#'} {'css', 'c#'}
```

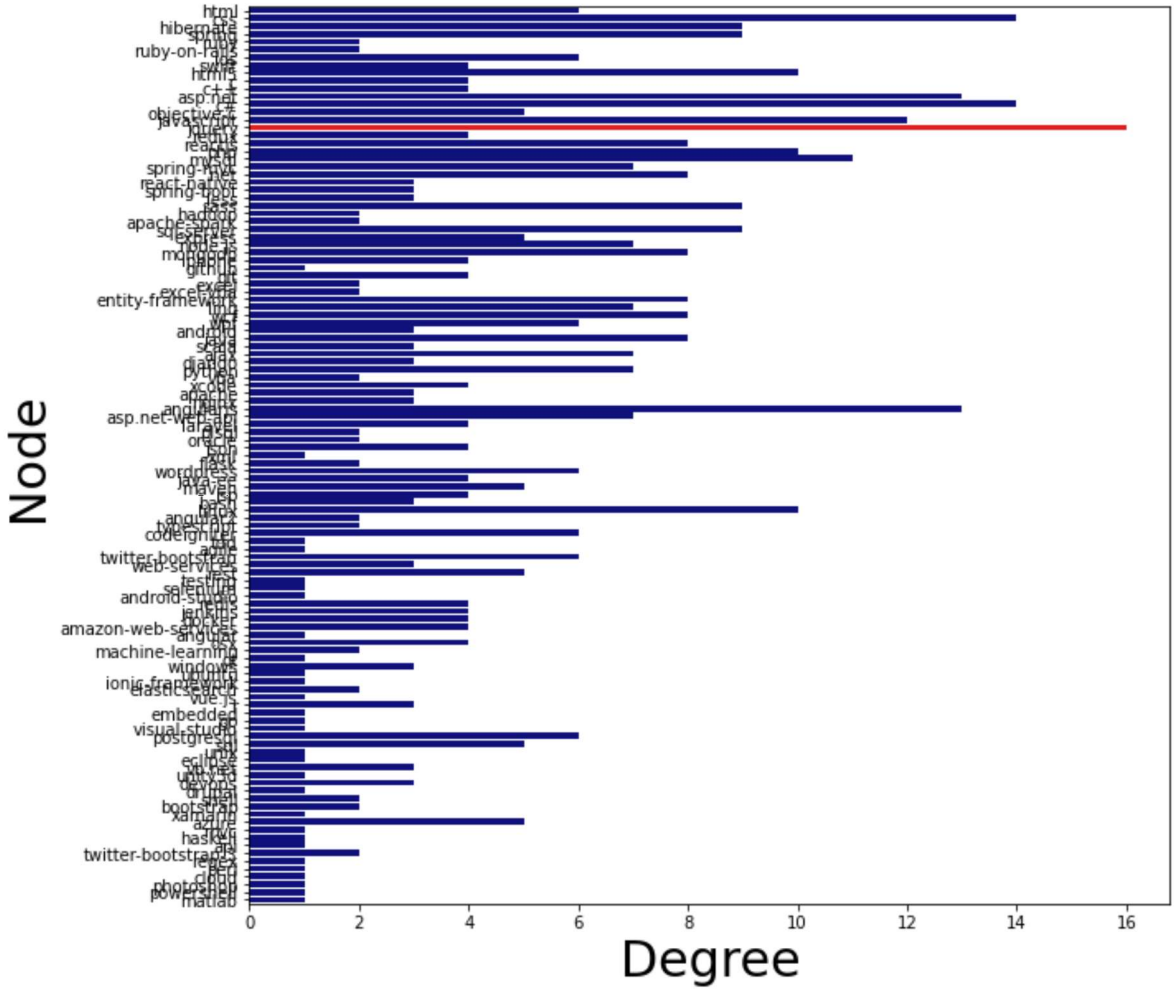
Degree Centrality

In [13]:

```
from collections import Counter
degree_dic = dict(G.degree(G.nodes()))
nx.set_node_attributes(G, degree_dic, 'degree')

degree_hist = pd.DataFrame({"degree": list(degree_dic.values()),
                            "Nodes": list(degree_dic.keys())})
plt.figure(figsize=(10,10))
clrs = ['darkblue' if (x < max(degree_dic.values())) else 'red' for x in degree_dic.values(
sns.barplot(x = 'degree', y = 'Nodes',
             data = degree_hist,
             palette=clrs)
plt.ylabel('Node', fontsize=30)
plt.xlabel('Degree', fontsize=30)
plt.tick_params(axis='both', which='major',labelsize=10)

plt.show()
plt.savefig('nodedegree.png')
# clearly 'jquery' has highest degree
```



```
<Figure size 432x288 with 0 Axes>
```

In [14]:

```
# degree of each node
degree=G.degree(G.nodes())
for i in degree:
    print(i,end=" ")
```

```
('html', 6) ('css', 14) ('hibernate', 9) ('spring', 9) ('ruby', 2) ('ruby-on
-rails', 2) ('ios', 6) ('swift', 4) ('html5', 10) ('c', 4) ('c++', 4) ('asp.
net', 13) ('c#', 14) ('objective-c', 5) ('javascript', 12) ('jquery', 16)
('redux', 4) ('reactjs', 8) ('php', 10) ('mysql', 11) ('spring-mvc', 7) ('.n
et', 8) ('react-native', 3) ('spring-boot', 3) ('less', 3) ('sass', 9) ('had
oop', 2) ('apache-spark', 2) ('sql-server', 9) ('express', 5) ('node.js', 7)
('mongodb', 8) ('iphone', 4) ('github', 1) ('git', 4) ('excel', 2) ('excel-v
ba', 2) ('entity-framework', 8) ('linq', 7) ('wcf', 8) ('wpf', 6) ('androi
d', 3) ('java', 8) ('scala', 3) ('ajax', 7) ('django', 3) ('python', 7) ('vb
a', 2) ('xcode', 4) ('apache', 3) ('nginx', 3) ('angularjs', 13) ('asp.net-w
eb-api', 7) ('laravel', 4) ('plsql', 2) ('oracle', 2) ('json', 4) ('xml', 1)
('flask', 2) ('wordpress', 6) ('java-ee', 4) ('maven', 5) ('jsp', 4) ('bas
h', 3) ('linux', 10) ('angular2', 2) ('typescript', 2) ('codeigniter', 6)
('tdd', 1) ('agile', 1) ('twitter-bootstrap', 6) ('web-services', 3) ('res
t', 5) ('testing', 1) ('selenium', 1) ('android-studio', 1) ('redis', 4) ('j
enkins', 4) ('docker', 4) ('amazon-web-services', 4) ('angular', 1) ('osx',
4) ('machine-learning', 2) ('qt', 1) ('windows', 3) ('ubuntu', 1) ('ionic-fr
amework', 1) ('elasticsearch', 2) ('vue.js', 1) ('r', 3) ('embedded', 1) ('g
o', 1) ('visual-studio', 1) ('postgresql', 6) ('sql', 5) ('unix', 1) ('eclip
se', 1) ('vb.net', 3) ('unity3d', 1) ('devops', 3) ('drupal', 1) ('shell',
2) ('bootstrap', 2) ('xamarin', 1) ('azure', 5) ('mvc', 1) ('haskell', 1)
('api', 1) ('twitter-bootstrap-3', 2) ('regex', 1) ('perl', 1) ('cloud', 1)
('photoshop', 1) ('powershell', 1) ('matlab', 1)
```

In [15]:

```
# Degree centrality
dc_dict = nx.degree_centrality(G)
nx.set_node_attributes(G, dc_dict, 'degree')
sorted_dc = sorted(dc_dict.items(), key=itemgetter(1), reverse=True)
print("Order of nodes According to their importance by using Degree centrality")
for i in sorted_dc:
    print(i)
```

```
Order of nodes According to their importance by using Degree centrality
('jquery', 0.14035087719298245)
('css', 0.12280701754385964)
('c#', 0.12280701754385964)
('asp.net', 0.11403508771929824)
('angularjs', 0.11403508771929824)
('javascript', 0.10526315789473684)
('mysql', 0.09649122807017543)
('html5', 0.08771929824561403)
('php', 0.08771929824561403)
('linux', 0.08771929824561403)
('hibernate', 0.07894736842105263)
('spring', 0.07894736842105263)
('sass', 0.07894736842105263)
('sql-server', 0.07894736842105263)
('reactjs', 0.07017543859649122)
('.net', 0.07017543859649122)
('mongodb', 0.07017543859649122)
('entity-framework', 0.07017543859649122)
```

In [16]:

```
# TOp 3 nodes with highest Degree centrality
print("\nTop three nodes having highest Degree centrality")
for i in sorted_dc[:3]:
    print(i)
```

```
Top three nodes having highest Degree centrality
('jquery', 0.14035087719298245)
('css', 0.12280701754385964)
('c#', 0.12280701754385964)
```

Eigenvector Centrality

In [17]:

```
# Eigenvector Centrality
eigenvector_dict = nx.eigenvector_centrality(G)
nx.set_node_attributes(G, eigenvector_dict, 'eigenvector')
sorted_eigenvector = sorted(eigenvector_dict.items(), key=itemgetter(1), reverse=True)
print("Order of nodes According to their importance by using EigenVector Centrality")
for i in sorted_eigenvector:
    print(i)
```

```
Order of nodes According to their importance by using EigenVector Centrali
ty
('jquery', 0.3657638453622554)
('css', 0.338701180241117)
('javascript', 0.32563098638889276)
('html5', 0.2681052746250041)
('php', 0.26530101525817973)
('angularjs', 0.2652026528173442)
('sass', 0.2520957761852705)
('mysql', 0.239342658253863)
('twitter-bootstrap', 0.20709455337131874)
('html', 0.2038246927757326)
('ajax', 0.19962166194538747)
('wordpress', 0.16028159729147778)
('codeigniter', 0.14895861770584937)
('reactjs', 0.13770631051407162)
('asp.net', 0.1133542478710743)
('node.js', 0.11189410352446867)
('mongodb', 0.1032689802929971)
```

In [18]:

```
# Top 3 nodes with highest Eigenvector centrality
print("\nTop three nodes having highest Eigenvector centrality")
for i in sorted_eigenvector[:3]:
    print(i)
```

```
Top three nodes having highest Eigenvector centrality
('jquery', 0.3657638453622554)
('css', 0.338701180241117)
('javascript', 0.32563098638889276)
```

Betweenness Centrality

In [19]:

```
# Betweenness Centrality
betweenness_dict = nx.betweenness_centrality(G)
nx.set_node_attributes(G, betweenness_dict, 'betweenness')
sorted_betweenness = sorted(betweenness_dict.items(), key=itemgetter(1), reverse=True)
print("Order of nodes According to their importance by using Betweenness Centrality")
for i in sorted_betweenness:
    print(i)
```

```
Order of nodes According to their importance by using Betweenness Centrali
ty
('jquery', 0.2555399753457234)
('linux', 0.20840160874161803)
('mysql', 0.1976931477327379)
('asp.net', 0.17406690608353667)
('apache', 0.13087186063431988)
('json', 0.12319763505138448)
('angularjs', 0.12286762799187795)
('rest', 0.11370118004957315)
('python', 0.11018306022497917)
('postgresql', 0.08761830339613291)
('java', 0.07601405167311084)
('mongodb', 0.06737252741910403)
('osx', 0.06635551651853466)
('jenkins', 0.05981851611101727)
('django', 0.05779082240190766)
('amazon-web-services', 0.056437373973470865)
('c#', 0.055756579881898374)
```

In [20]:

```
# Top 3 nodes with highest Betweenness centrality
print("\nTop three nodes having highest Betweenness centrality")
for b in sorted_betweenness[:3]:
    print(b)
```

```
Top three nodes having highest Betweenness centrality
('jquery', 0.2555399753457234)
('linux', 0.20840160874161803)
('mysql', 0.1976931477327379)
```

In [21]:

```
# shortest path between nodes
nx.shortest_path(G,'jquery','c#')
```

Out[21]:

```
['jquery', 'asp.net', 'c#']
```

In [22]:

```
nx.shortest_path(G,'jquery','redux')
```

Out[22]:

```
['jquery', 'angularjs', 'express', 'redux']
```

In [23]:

```
nx.shortest_path(G,'linq','xml')
```

Out[23]:

```
['linq', 'asp.net', 'jquery', 'json', 'xml']
```

In [24]:

```
# from all the 3 centralities common node is 'jquery'
# since it is not connected
# shortest path lengths from highest centrality node 'jquery' are
shortlength_jquery=nx.shortest_path_length(G,'jquery')
shortlength_jquery
```

Out[24]:


```
{'jquery': 0,
 'html': 1,
 'twitter-bootstrap-3': 1,
 'twitter-bootstrap': 1,
 'angularjs': 1,
 'css': 1,
 'json': 1,
 'php': 1,
 'mysql': 1,
 'ajax': 1,
 'sass': 1,
 'codeigniter': 1,
 'javascript': 1,
 'html5': 1,
 'bootstrap': 1,
 'wordpress': 1,
 'asp.net': 1,
 'linq': 2,
 'postgresql': 2,
 'sql': 2,
 'xml': 2,
 'laravel': 2,
 'angular2': 2,
 'ionic-framework': 2,
 'rest': 2,
 'drupal': 2,
 'node.js': 2,
 'reactjs': 2,
 'vb.net': 2,
 'mongodb': 2,
 '.net': 2,
 'photoshop': 2,
 'azure': 2,
 'apache': 2,
 'c#': 2,
 'less': 2,
 'wpf': 2,
 'entity-framework': 2,
 'wcf': 2,
 'mvc': 2,
 'express': 2,
 'asp.net-web-api': 2,
 'sql-server': 2,
 'elasticsearch': 3,
 'xamarin': 3,
 'visual-studio': 3,
 'typescript': 3,
 'nginx': 3,
 'redis': 3,
 'oracle': 3,
 'unity3d': 3,
```

```
'spring': 3,
'plsql': 3,
'linux': 3,
'ruby-on-rails': 3,
'amazon-web-services': 3,
'vue.js': 3,
'ruby': 3,
'web-services': 3,
'react-native': 3,
'redux': 3,
'api': 3,
'hibernate': 3,
'django': 3,
'windows': 4,
'angular': 4,
'unix': 4,
'java-ee': 4,
'osx': 4,
'spring-boot': 4,
'docker': 4,
'cloud': 4,
'git': 4,
'shell': 4,
'python': 4,
'spring-mvc': 4,
'flask': 4,
'maven': 4,
'jsp': 4,
'java': 4,
'ubuntu': 4,
'devops': 4,
'bash': 4,
'jenkins': 5,
'go': 5,
'ios': 5,
'objective-c': 5,
'eclipse': 5,
'c': 5,
'c++': 5,
'powershell': 5,
'r': 5,
'android': 5,
'machine-learning': 5,
'github': 5,
'xcode': 6,
'qt': 6,
'embedded': 6,
'matlab': 6,
'swift': 6,
'iphone': 6,
'android-studio': 6}
```

Building a subgroup

- We can find the distance of a node from every other node in the network using breadth-first search algorithm, starting from that node. networkX provides the function bfs_tree to do it.

In [25]:

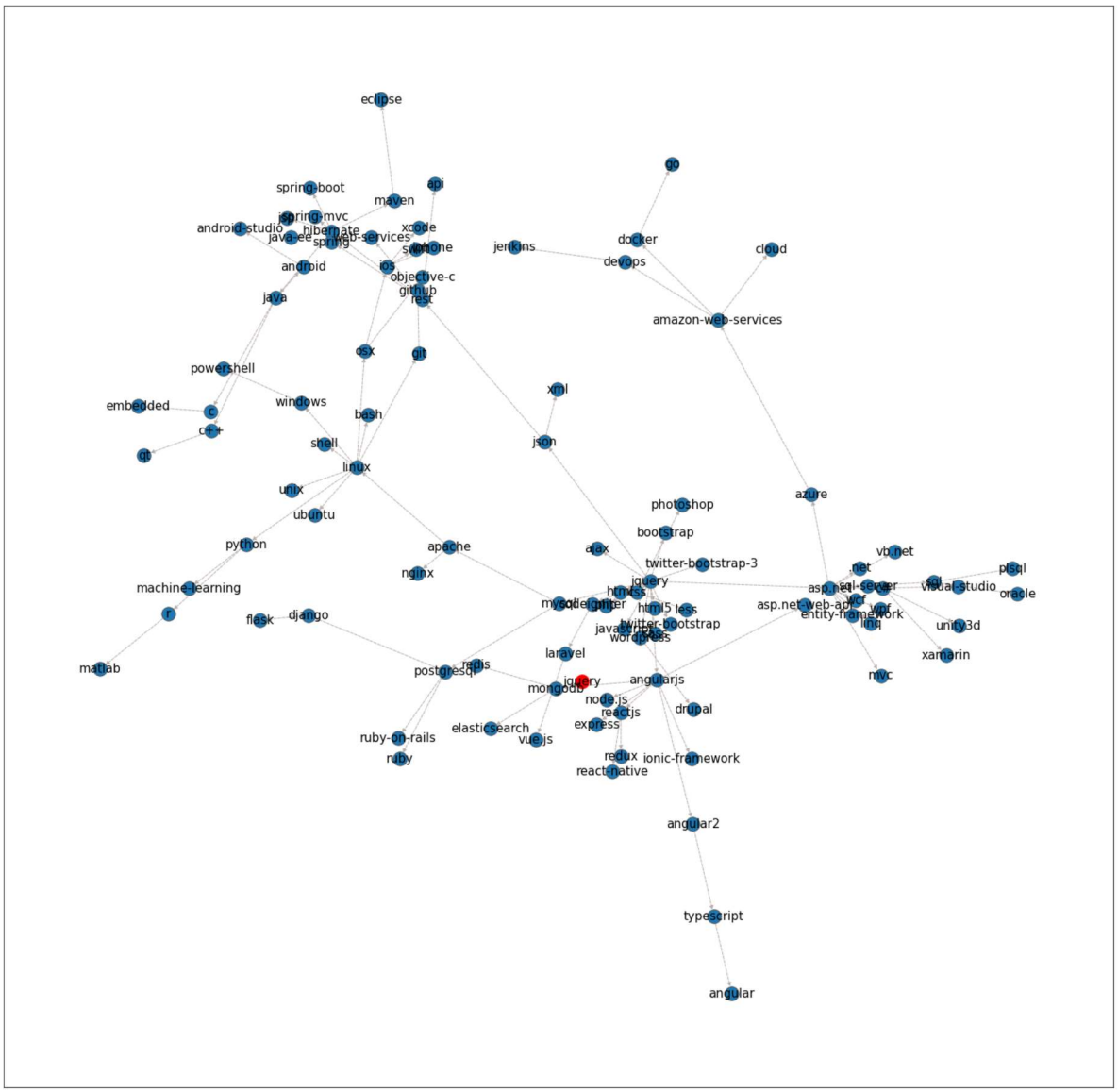
```
sub1 = nx.bfs_tree(G,'jquery')
```

In [26]:

```
sub2 = nx.bfs_tree(G,'css')
```

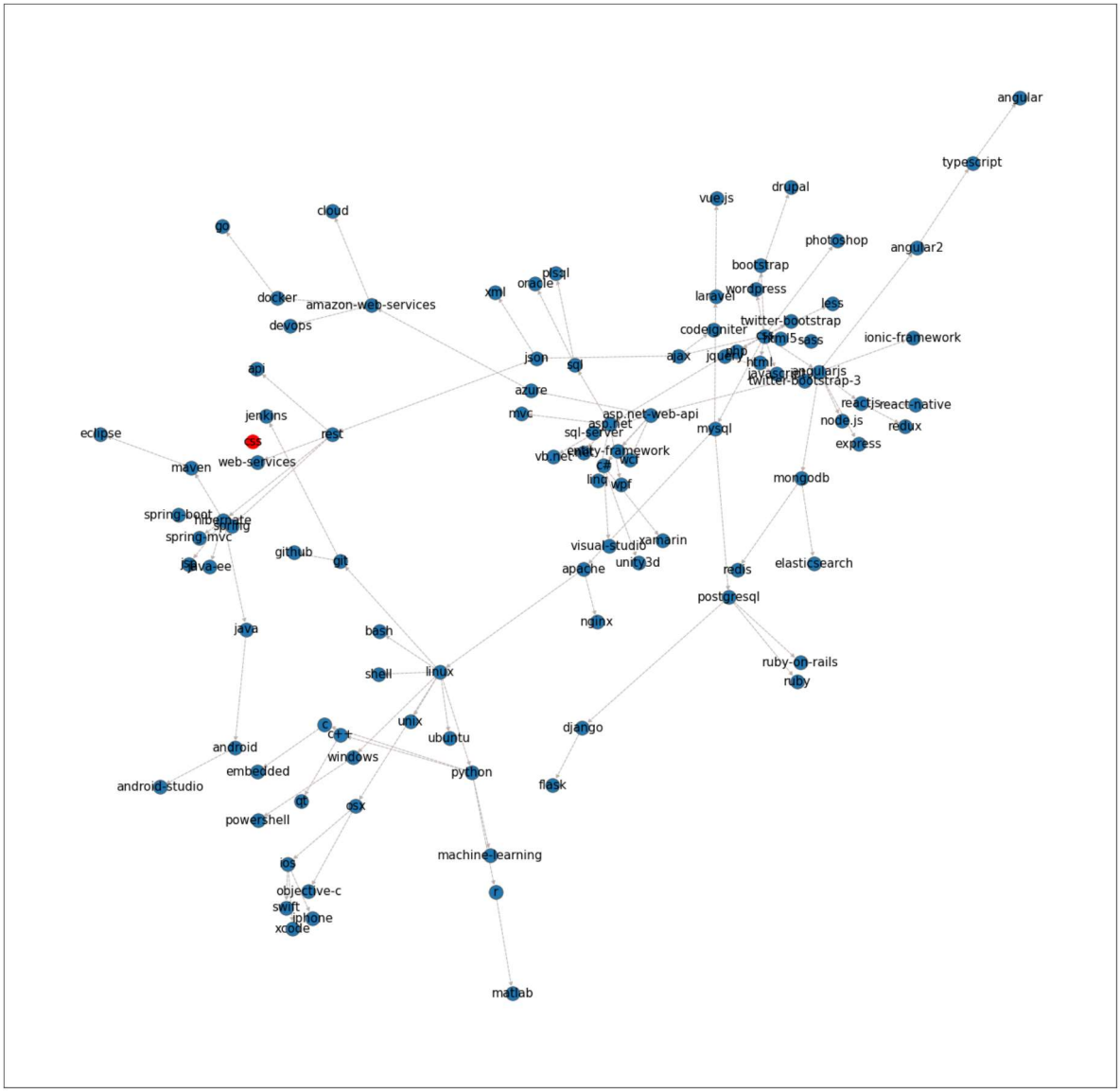
In [27]:

```
# Subgroup (an oriented tree constructed from of a breadth-first-search starting at "jquery
plt.figure(figsize=(25, 25))
options = {
    'edge_color': '#BAB0AD',
    'width': 1,
    'with_labels': True,
    'font_weight': 'normal',
    'font_size': 15,
    'style': 'dashed'
}
sizes = [G.nodes[node]['nodesize'] * 10 for node in G]
nx.draw_networkx(sub1, pos=nx.spring_layout(G, k=0.25, iterations=50), **options)
nx.draw_networkx(sub1.subgraph('jquery'), pos=nx.spring_layout(G, k=0.25, iterations=50),no
ax = plt.gca()
ax.collections[0].set_edgecolor("#555555")
plt.show()
```



In [28]:

```
# Subgroup (an oriented tree constructed from of a breadth-first-search starting at "css")
plt.figure(figsize=(25, 25))
options = {
    'edge_color': '#BAB0AD',
    'width': 1,
    'with_labels': True,
    'font_weight': 'normal',
    'font_size': 15,
    'style': 'dashed'
}
sizes = [G.nodes[node]['nodesize'] * 10 for node in G]
nx.draw_networkx(sub2, pos=nx.spring_layout(G, k=0.25, iterations=50), **options)
nx.draw_networkx(sub2.subgraph('css'), pos=nx.spring_layout(G, k=0.25, iterations=50),node_
ax = plt.gca()
ax.collections[0].set_edgecolor("#555555")
plt.show()
```



In [29]:

```
# Nodes which are connected to important node
print(nx.node_connected_component(G,'jquery'))
print("\nTotal {} nodes are connected with main important 'jquery' node".format(len(nx.node
```

```
{'elasticsearch', 'postgresql', 'visual-studio', 'typescript', 'laravel', 't
witter-bootstrap-3', 'embedded', 'php', 'vb.net', 'reactjs', 'swift', 'sprin
g-boot', 'amazon-web-services', 'azure', 'ajax', 'apache', 'less', 'python',
'spring-mvc', 'wpf', 'machine-learning', 'flask', 'java', 'mvc', 'redux', 'd
jango', 'jenkins', 'twitter-bootstrap', 'nginx', 'sass', 'objective-c', 'ang
ular2', 'unix', 'spring', 'vue.js', 'git', 'r', 'bootstrap', 'android', 'web
-services', 'entity-framework', 'maven', 'devops', 'bash', 'asp.net-web-ap
i', 'eclipse', 'asp.net', 'xcode', 'windows', 'ios', 'redis', 'ionic-framewo
rk', 'unity3d', 'matlab', 'plsql', 'linux', 'javascript', 'node.js', 'mongod
b', '.net', 'iphone', 'docker', 'c++', 'powershell', 'android-studio', 'cs
s', 'wordpress', 'shell', 'angularjs', 'mysql', 'linq', 'xamarin', 'go', 'sq
l', 'xml', 'codeigniter', 'oracle', 'angular', 'qt', 'html', 'java-ee', 'res
t', 'json', 'osx', 'drupal', 'html5', 'c', 'ruby-on-rails', 'photoshop', 'cl
oud', 'c#', 'ruby', 'jquery', 'react-native', 'github', 'jsp', 'wcf', 'ubunt
u', 'api', 'express', 'hibernate', 'sql-server'}

Total 102 nodes are connected with main important 'jquery' node
```

In [30]:

```
# Nodes which are not connected to important node
print(G.nodes()-list(nx.node_connected_component(G,'jquery')))
```

```
{'apache-spark', 'hadoop', 'selenium', 'vba', 'haskell', 'perl', 'excel', 'e
xcel-vba', 'regex', 'testing', 'agile', 'tdd', 'scala'}
```

Node Classification

In [31]:

```
# harmonic_function
G.nodes['angular']['label']='web API framework'
G.nodes['css']['label']='web design '
G.nodes['c++']['label']='programming language'
G.nodes['git']['label']='command shell'
G.nodes['linux']['label']='OS'
G.nodes['qt']['label']='GUI'
G.nodes['hibernate']['label']='database'
classs = node_classification.harmonic_function(G)
nodes=list(G.nodes())
node_class={nodes[i]:classs[i] for i in range(len(nodes))}
for i in node_class:
    print(i,"-------->",node_class.get(i))
```

```
html --------> web design
css --------> web design
hibernate --------> database
spring --------> database
ruby --------> web design
ruby-on-rails --------> web design
ios --------> OS
swift --------> OS
html5 --------> web design
c --------> programming language
c++ --------> programming language
asp.net --------> web design
c# --------> web design
objective-c --------> OS
javascript --------> web design
jquery --------> web design
redux --------> web design
reactjs --------> web design
php --------> web design
```

Link prediction

- Jaccard Coefficient

In [32]:

```
# jaccard coefficient
threshold_j=0.45
jaccard=list(nx.jaccard_coefficient(G))
for i in jaccard:
    if i[2]>threshold_j:
        print(i)
```

```
('visual-studio', 'unity3d', 1.0)
('visual-studio', 'xamarin', 1.0)
('vb.net', 'sql', 0.6)
('spring-boot', 'java-ee', 0.75)
('spring-boot', 'web-services', 0.5)
('spring-boot', 'maven', 0.6)
('spring-boot', 'jsp', 0.75)
('ajax', 'html', 0.625)
('less', 'twitter-bootstrap', 0.5)
('wpf', 'sql-server', 0.6666666666666666)
('machine-learning', 'matlab', 0.5)
('twitter-bootstrap', 'html', 0.5)
('angular2', 'ionic-framework', 0.5)
('angular2', 'angular', 0.5)
('unix', 'shell', 0.5)
('unix', 'ubuntu', 1.0)
('git', 'shell', 0.5)
('bootstrap', 'photoshop', 0.5)
('maven', 'java-ee', 0.5)
```

- Resource Allocation Index (predict missing links, similarity between two nodes)

In [33]:

```
# Resource Allocation Index
threshold_RAI=0.45
RAI=list(nx.resource_allocation_index(G))
for i in RAI:
    if i[2]>threshold_RAI:
        print(i)
```

```
('typescript', 'angularjs', 0.5)
('php', 'sass', 0.48392857142857143)
('amazon-web-services', 'jenkins', 0.5833333333333333)
('python', 'java', 0.5)
('wpf', 'sql-server', 0.6662087912087913)
('redux', 'mongodb', 0.46785714285714286)
('redux', 'angularjs', 0.46785714285714286)
('angular2', 'angular', 0.5)
('asp.net-web-api', 'linq', 0.5094627594627594)
('asp.net-web-api', '.net', 0.7094627594627594)
('css', 'codeigniter', 0.5629329004329003)
('java-ee', 'jsp', 0.49007936507936506)
('react-native', 'express', 0.5178571428571428)
```

- Adamic Adar Index

1. predict missing links in a Network, according to the amount of shared links between two nodes

In [34]:

```
# Adamic adar Index
threshold_AAI=1.45
AAI=list(nx.adamic_adar_index(G))
for i in AAI:
    if i[2]>threshold_AAI:
        print(i)
```

```
('php', 'angularjs', 1.5763210281972886)
('php', 'twitter-bootstrap', 1.5763210281972886)
('php', 'sass', 2.1344316547485356)
('amazon-web-services', 'jenkins', 1.631586747071319)
('azure', 'entity-framework', 1.76359111627397)
('azure', 'wcf', 1.76359111627397)
('azure', 'sql-server', 1.76359111627397)
('ajax', 'wordpress', 1.732002050366691)
('ajax', 'html', 1.9933534196215348)
('ajax', 'html5', 1.5763210281972886)
('wpf', 'asp.net-web-api', 1.730591120867207)
('wpf', 'sql-server', 2.725387810199946)
('redux', 'mongodb', 1.6161316238923504)
('redux', 'angularjs', 1.6161316238923504)
('twitter-bootstrap', 'html', 1.5971461596074552)
('sass', 'mysql', 1.700137172845284)
('asp.net-web-api', 'linq', 2.185710734180626)
('asp.net-web-api', '.net', 2.807045668740238)
('javascript', 'codeigniter', 1.7258989759194896)
('javascript', 'mongodb', 1.8017003260082651)
('javascript', 'wordpress', 1.6081859057186956)
('css', 'codeigniter', 2.284009602470737)
('wordpress', 'mysql', 1.732002050366691)
('angularjs', 'html', 1.5971461596074552)
('angularjs', 'mysql', 1.6229248932570246)
('mysql', 'html5', 1.5763210281972884)
('html', 'html5', 2.031440641510707)
('java-ee', 'jsp', 1.9050359159595758)
('react-native', 'express', 1.7161442097772204)
```

- Preferential Attachment

1. The probability of generating a new link of node u is directly proportional to the degree of the node

In [35]:

```
#preferential Attachment
degree_dic = dict(G.degree(G.nodes()))
minn=min(degree_dic.values())
maxx=max(degree_dic.values())
avg=sum(degree_dic.values())/len(degree_dic)
print("Minimum degree = {}\nMaximum degree={}\nAverage Degree={}".format(minn,maxx,avg))
```

```
Minimum degree = 1
Maximum degree=16
Average Degree=4.260869565217392
```

In [36]:

```
thres_d=maxx*avg
PA=list(nx.preferential_attachment(G))
for i in PA:
    if i[2]>thres_d:
        print(i)
```

```
('postgresql', 'javascript', 72)
('postgresql', 'css', 84)
('postgresql', 'angularjs', 78)
('postgresql', 'c#', 84)
('postgresql', 'jquery', 96)
('postgresql', 'asp.net', 78)
('php', 'linux', 100)
('php', 'node.js', 70)
('php', 'reactjs', 80)
('php', 'mongodb', 80)
('php', '.net', 80)
('php', 'python', 70)
('php', 'spring-mvc', 70)
('php', 'angularjs', 130)
('php', 'java', 80)
('php', 'linq', 70)
('php', 'sass', 90)
('php', 'spring', 90)
('php', 'c#', 140)
```

Community Detection

- Greedy Modularity Community

In [37]:

```
# Greedy Modularity
communities = community.greedy_modularity_communities(G)
for i in communities:
    print(i,"\n")
```

```
frozenset({'elasticsearch', 'visual-studio', 'postgresql', 'typescript', 'la
ravel', 'twitter-bootstrap-3', 'embedded', 'php', 'vb.net', 'reactjs', 'swif
t', 'spring-boot', 'amazon-web-services', 'azure', 'ajax', 'apache', 'less',
'python', 'spring-mvc', 'wpf', 'machine-learning', 'flask', 'java', 'mvc',
'redux', 'django', 'jenkins', 'twitter-bootstrap', 'nginx', 'sass', 'objecti
ve-c', 'angular2', 'unix', 'spring', 'vue.js', 'git', 'r', 'android', 'boots
trap', 'web-services', 'entity-framework', 'maven', 'devops', 'bash', 'asp.n
et-web-api', 'eclipse', 'asp.net', 'xcode', 'windows', 'ios', 'redis', 'ioni
c-framework', 'unity3d', 'matlab', 'plsql', 'linux', 'javascript', 'node.j
s', 'mongodb', '.net', 'iphone', 'c++', 'docker', 'powershell', 'android-stu
dio', 'css', 'wordpress', 'shell', 'angularjs', 'mysql', 'linq', 'xamarin',
'go', 'sql', 'xml', 'codeigniter', 'oracle', 'angular', 'qt', 'html', 'java-
ee', 'rest', 'json', 'osx', 'drupal', 'c', 'html5', 'ruby-on-rails', 'photos
hop', 'cloud', 'c#', 'ruby', 'jquery', 'react-native', 'github', 'jsp', 'wc
f', 'ubuntu', 'api', 'express', 'hibernate', 'sql-server'})

frozenset({'apache-spark', 'hadoop', 'haskell', 'scala'})

frozenset({'excel', 'excel-vba', 'vba'})

frozenset({'tdd', 'agile'})

frozenset({'selenium', 'testing'})

frozenset({'regex', 'perl'})
```

In [38]:

```
modularity_dict = {} # Create a blank dictionary
for i,c in enumerate(communities): # Loop through the list of communities, keeping track of
    for name in c: # Loop through each person in a community
        modularity_dict[name] = i # Create an entry in the dictionary for the person, where

# Now you can add modularity information like we did the other metrics
nx.set_node_attributes(G, modularity_dict, 'modularity')
```

In [39]:

```
# First get a list of just the nodes in that class
class0 = [n for n in G.nodes() if G.nodes[n]['modularity'] == 0]

# Then create a dictionary of the eigenvector centralities of those nodes
class0_eigenvector = {n:G.nodes[n]['eigenvector'] for n in class0}

# Then sort that dictionary and print the first 5 results
class0_sorted_by_eigenvector = sorted(class0_eigenvector.items(), key=itemgetter(1), revers

print("Modularity Class 0 Sorted by Eigenvector Centrality:")
for node in class0_sorted_by_eigenvector[:5]:
    print("Name:", node[0], "| Eigenvector Centrality:", node[1])
```

```
Modularity Class 0 Sorted by Eigenvector Centrality:
Name: jquery | Eigenvector Centrality: 0.3657638453622554
Name: css | Eigenvector Centrality: 0.338701180241117
Name: javascript | Eigenvector Centrality: 0.32563098638889276
Name: html5 | Eigenvector Centrality: 0.2681052746250041
Name: php | Eigenvector Centrality: 0.26530101525817973
```

In [40]:

```
for i,c in enumerate(communities): # Loop through the list of communities
    if len(c) > 2: # Filter out modularity classes with 2 or fewer nodes
        print('Class '+str(i)+':', list(c),"\n") # Print out the classes and their members
```

```
Class 0: ['elasticsearch', 'visual-studio', 'postgresql', 'typescript', 'lar
avel', 'twitter-bootstrap-3', 'embedded', 'php', 'vb.net', 'reactjs', 'swif
t', 'spring-boot', 'amazon-web-services', 'azure', 'ajax', 'apache', 'less',
'python', 'spring-mvc', 'wpf', 'machine-learning', 'flask', 'java', 'mvc',
'redux', 'django', 'jenkins', 'twitter-bootstrap', 'nginx', 'sass', 'objecti
ve-c', 'angular2', 'unix', 'spring', 'vue.js', 'git', 'r', 'android', 'boots
trap', 'web-services', 'entity-framework', 'maven', 'devops', 'bash', 'asp.n
et-web-api', 'eclipse', 'asp.net', 'xcode', 'windows', 'ios', 'redis', 'ioni
c-framework', 'unity3d', 'matlab', 'plsql', 'linux', 'javascript', 'node.j
s', 'mongodb', '.net', 'iphone', 'c++', 'docker', 'powershell', 'android-stu
dio', 'css', 'wordpress', 'shell', 'angularjs', 'mysql', 'linq', 'xamarin',
'go', 'sql', 'xml', 'codeigniter', 'oracle', 'angular', 'qt', 'html', 'java-
ee', 'rest', 'json', 'osx', 'drupal', 'c', 'html5', 'ruby-on-rails', 'photos
hop', 'cloud', 'c#', 'ruby', 'jquery', 'react-native', 'github', 'jsp', 'wc
f', 'ubuntu', 'api', 'express', 'hibernate', 'sql-server']

Class 1: ['apache-spark', 'hadoop', 'haskell', 'scala']

Class 2: ['excel', 'excel-vba', 'vba']
```

- label_propagation_communities

In [41]:

```
# Label Propagation communities
label_prop_comm=community.label_propagation_communities(G)
j=0
for i in label_prop_comm:
    print("Class [{}] ----> {}".format(j,i))
    j+=1
```

```
Class [0] ----> {'c++', 'qt'}
Class [1] ----> {'android', 'android-studio'}
Class [2] ----> {'flask', 'django', 'python'}
Class [3] ----> {'excel', 'excel-vba', 'vba'}
Class [4] ----> {'mongodb', 'express', 'node.js', 'redux', 'reactjs', 'react
-native'}
Class [5] ----> {'angular2', 'angular', 'typescript'}
Class [6] ----> {'tdd', 'agile'}
Class [7] ----> {'spring-boot', 'java-ee', 'rest', 'maven', 'jsp', 'java',
'spring', 'web-services', 'api', 'hibernate', 'eclipse', 'spring-mvc'}
Class [8] ----> {'selenium', 'testing'}
Class [9] ----> {'redis', 'elasticsearch', 'postgresql', 'ruby', 'ruby-on-ra
ils'}
Class [10] ----> {'iphone', 'xcode', 'ios', 'osx', 'objective-c', 'swift'}
Class [11] ----> {'windows', 'powershell'}
Class [12] ----> {'c', 'embedded'}
Class [13] ----> {'oracle', 'plsql'}
Class [14] ----> {'jenkins', 'amazon-web-services', 'go', 'docker', 'devop
s', 'cloud'}
Class [15] ----> {'github', 'ubuntu', 'linux', 'nginx', 'bash', 'apache', 'g
it', 'shell', 'unix'}
Class [16] ----> {'entity-framework', '.net', 'linq', 'unity3d', 'xamarin',
'visual-studio', 'wcf', 'sql', 'azure', 'mvc', 'c#', 'asp.net-web-api', 'vb.
net', 'sql-server', 'wpf', 'asp.net'}
Class [17] ----> {'apache-spark', 'hadoop', 'haskell', 'scala'}
Class [18] ----> {'twitter-bootstrap', 'sass', 'xml', 'laravel', 'codeignite
r', 'ionic-framework', 'html', 'twitter-bootstrap-3', 'json', 'php', 'drupa
l', 'javascript', 'html5', 'photoshop', 'vue.js', 'ajax', 'less', 'css', 'wo
rdpress', 'jquery', 'bootstrap', 'angularjs', 'mysql'}
Class [19] ----> {'regex', 'perl'}
Class [20] ----> {'r', 'matlab', 'machine-learning'}
```

Drawing Networks

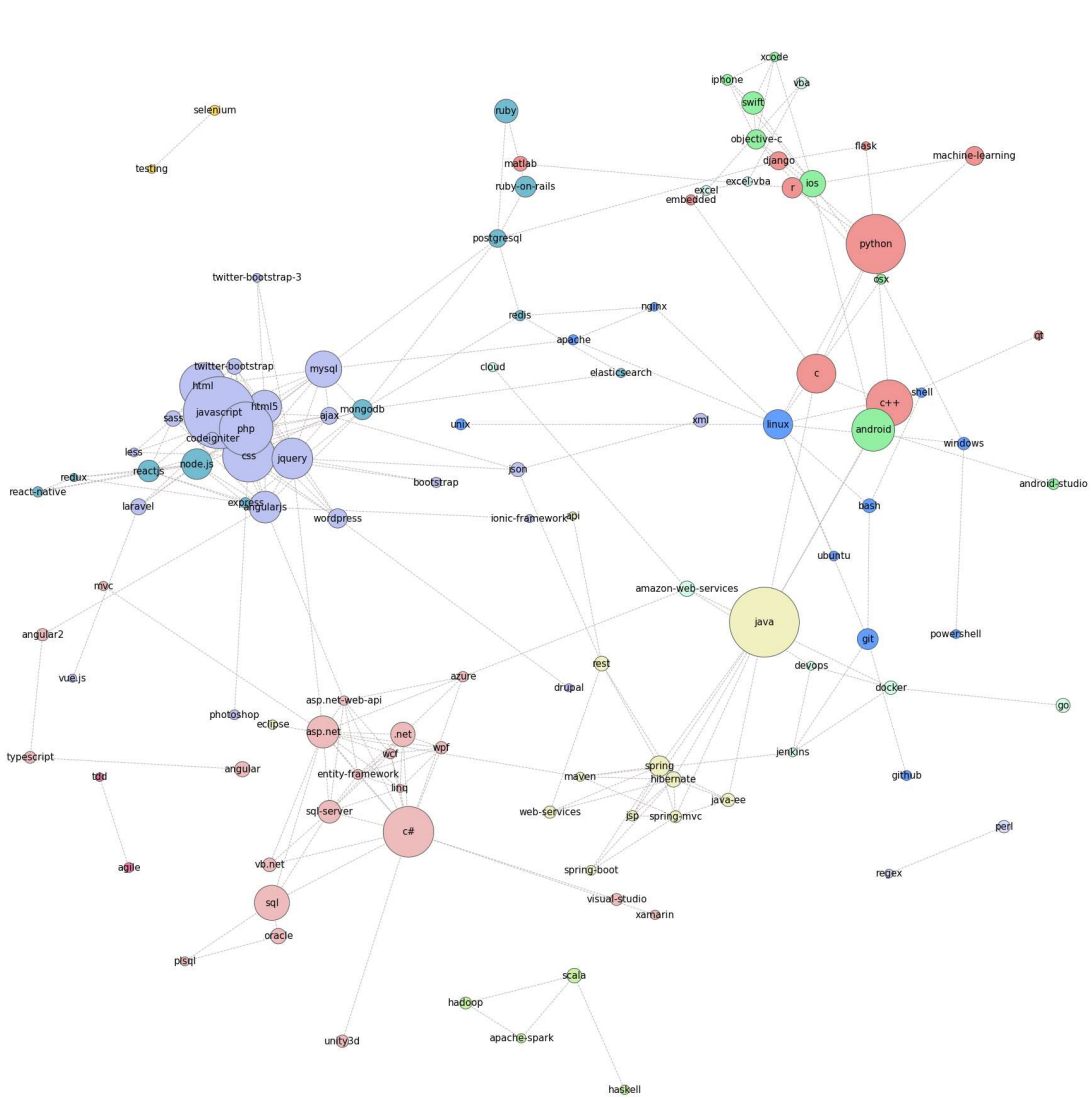
In [42]:

```
color_map = {1: '#f09494', 2: '#eebcbc', 3: '#72bbd0', 4: '#91f0a1', 5: '#629fff', 6: '#bcc
             7: '#eebcbc', 8: '#f1f0c0', 9: '#d2ffe7', 10: '#caf3a6', 11: '#ffdf55', 12: '#
             13: '#d6dcff', 14: '#d2f5f0', 15: '#2B2B40', 16: '#e6bbaa', 17: '#c158fd'}

plt.figure(figsize=(25, 25))
options = {
    'edge_color': '#BAB0AD',
    'width': 1,
    'with_labels': True,
    'font_weight': 'normal',
    'font_size': 15,
    'style': 'dashed'
}
colors = [color_map[G.nodes[node]['group']] for node in G]
sizes = [G.nodes[node]['nodesize'] * 20 for node in G]

"""
Using the spring layout :
- k controls the distance between the nodes and varies between 0 and 1
- iterations is the number of times simulated annealing is run
default k=0.1 and iterations=50
"""

# nx.spring_layout(G, k=0.25, iterations=50)
nx.draw(G, node_color=colors, node_size=sizes, pos=nx.spring_layout(G, k=1, iterations=50),
ax = plt.gca()
ax.collections[0].set_edgecolor("#555555")
plt.show()
plt.savefig("Network.png")
```



<Figure size 432x288 with 0 Axes>