Code Frisk

Generated by Doxygen 1.8.20

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

Namespace Documentation

3.1 models Namespace Reference

Classes

- class add_data

Functions

- def remove_redundant_functions (content)
- def visualizer (list_of_files, similarity_matrix)
- def remove_macros (file_content)
- def remove_comments (file_content)
- def remove_comments_pythonfile (file_content)
- def preprocessing (list_of_paths, list_of_files)
- def tf_idf (word_count_in_each_file, word_count_across_documents, list_of_paths, list_of_files)
- def txt_file (similarity_matrix, list_of_paths, list_of_files)
- def csv_file (list_of_files, similarity_matrix)
- def similarity (s, t)

3.1.1 Function Documentation

3.1.1.1 csv_file()

```
def models.csv_file (
              list_of_files,
              similarity_matrix )


Arguments    :
    list_of_files    :list of source code file names
    similarity_matrix:2-dimensional matrix representing mutual similarity between each pair of files
Functionality:
    Plotting the output similarity_matrix and saving it as an csv file
```

```
304 def csv_file(list_of_files,similarity_matrix):
305     #""" Interpreting the Output data as a CSV file ,
306     #where each element represent the percentage matching between the file
307     #corresponding to a row and column"""
308     """
309     Arguments    :
310         list_of_files    :list of source code file names
311         similarity_matrix:2-dimensional matrix representing mutual similarity between each pair of files
312     Functionality:
313         Plotting the output similarity_matrix and saving it as an csv file
314 
315     """
316     f=similarity_matrix.tolist()
317     files=['']+list_of_files
318     for x in range(len(list_of_files)):
319         f[x]=[list_of_files[x]]+f[x]
320     f=[files]+f
321     with open("media/result.csv", "w+") as myCsv:
322         csvWriter = csv.writer(myCsv, delimiter=',')
323         csvWriter.writerows(f)
324     visualizer(list_of_files,similarity_matrix)
```

3.1.1.2 preprocessing()

```
def models.preprocessing (
              list_of_paths,
              list_of_files )


Core logic is based on the Bag_of_Words
and TF-IDF Strategy( Term frequency and Inverse Document Frequency )

Arguments    :
    list_of_paths    :list of source code paths
    list_of_files    :it consists of list of file names
Functionality:
    It finds the count of each word after removing comments and replacing macros and passes this vector to tf_
```

```
185 def preprocessing(list_of_paths,list_of_files):
186     """
187     Core logic is based on the Bag_of_Words
188     and TF-IDF Strategy( Term frequency and Inverse Document Frequency )
189 
190     Arguments    :
191         list_of_paths    :list of source code paths
192         list_of_files    :it consists of list of file names
193     Functionality:
194         It finds the count of each word after removing comments and replacing macros and passes this
      vector to tf_idf function.
195 
196     """
197     word_count_across_documents={}
198     """ Maintains the word count of each element in a document """
199 
200     word_count_in_each_file=[]
201 
202     for files in list_of_paths:
203 
204         filename='media/'+files
205         temp={}
206         myfile=open(filename,"r")
207         content=myfile.read()
208         myfile.close()
209 
210         if(files[-4:]=='.cpp'):
211             content=remove_comments(content)
212             content=remove_redundant_functions(content)
213             content=remove_macros(content)
214 
215         elif(files[-3:]=='.py'):
216             content=remove_comments_pythonfile(content)
217 
218 
219 
      sym=[",",";","{","}",")","(","[","]","+","-","*","/","%","|","&","^","!","=","<",">","?","'",'"','#','.']
220         for i in sym:
```

```
221             content=content.replace(i," "+i+" ")
222
223         if(files[-4:]=='.cpp'):
224             content=content.replace("while","for")
225             content=content.replace("switch","if")
226             content=content.replace("case","else if")
227             content=content.replace("default","else")
228             content=content.replace("do","")
229             content=content.replace(";","")
230             content=content.replace(",","")
231             content=content.replace("'","")
232             content=content.replace('"',"")
233
234         elif(files[-3:]=='.py'):
235             content=content.replace('while','for')
236             content=content.replace('switch','if')
237             content=content.replace('case','elif')
238             content=content.replace('default','else')
239             content=content.replace('do',"")
240             content=re.sub(':|\'|\"',",content)
241         while(content.find('/*')!=-1):
242             i=content.find('/*')
243             j=content.find('*/')
244             content=content[0:i]+content[j+2:]
245         List_of_words=content.split()
246
247         for i in List_of_words:
248             temp[i]=temp.get(i,0)+1
249             word_count_across_documents[i]=word_count_across_documents.get(i,0)+1
250
251         word_count_in_each_file.append(temp)
252
253     tf_idf(word_count_in_each_file,word_count_across_documents,list_of_paths,list_of_files)
254
```

3.1.1.3 remove_comments()

```
def models.remove_comments (
                file_content )
```

```
Arguments:
    file_content: string storing the source code
Return type: updated string
Functionality:
    All commments in the code are replaced.
Logic Used:
    Using Regex detect substrings starting with // and ending with \n
    Similarly detect substrings starting with /* and ending with */
```

```
153 def remove_comments(file_content):
154     """
155     Arguments:
156         file_content: string storing the source code
157     Return type: updated string
158     Functionality:
159         All commments in the code are replaced.
160     Logic Used:
161         Using Regex detect substrings starting with // and ending with \n
162         Similarly detect substrings starting with /* and ending with */
163     """
164     pattern=re.compile('//.*?$|/\*.*?\*/',re.DOTALL|re.MULTILINE)
165     """ pattern for comments """
166     content=re.sub(pattern,",file_content)
167     """remove comments"""
168     return content
169
```

3.1.1.4 remove_comments_pythonfile()

```
def models.remove_comments_pythonfile (
              file_content )
```

```
Arguments:
    file_content: string storing the source code in python
Return type: updated string
Functionality:
    All commments in the code are replaced.
Logic Used:
    Using Regex detect substrings starting with # and ending with \n
    Similarly detect substrings starting with ''' and ending with '''
```

```
170 def remove_comments_pythonfile(file_content):
171     """
172     Arguments:
173         file_content: string storing the source code in python
174     Return type: updated string
175     Functionality:
176         All commments in the code are replaced.
177     Logic Used:
178         Using Regex detect substrings starting with # and ending with \n
179         Similarly detect substrings starting with "' and ending with "'
180     """
181     pattern=re.compile('#.*?$|\'\'\'.*?\'\'\'|\"\"\".*?\"\"\"',re.DOTALL|re.MULTILINE)
182     content=re.sub(pattern,'',file_content)
183     return content
184
```

3.1.1.5 remove_macros()

```
def models.remove_macros (
              file_content )
```

```
Arguments:
    file_content: string storing the source code
Return type: updated string
Functionality:
    All macros in the code are replaced.
Logic Used:
    Using Regex detect the macros
    Replace them using replace() function
```

```
97 def remove_macros(file_content):
98
99     """
100     Arguments:
101         file_content: string storing the source code
102     Return type: updated string
103     Functionality:
104         All macros in the code are replaced.
105     Logic Used:
106         Using Regex detect the macros
107         Replace them using replace() function
108     """
109     m=re.findall('#define .+ .+',file_content)
110     """finding macros"""
111
112     content=re.sub('#define .+ .+','',file_content)
113     """removing macros definitions"""
114
115     for i in range(len(m)):
116         """replacing macros"""
117         j=0
118         n=0
119         for k in range(8,len(m[i])):
120             if(m[i][k]=='('):
121                 j=k-8
122                 break
```

```
123             elif(m[i][k]==' '):
124                 j=-1
125                 n=k+1
126                 break
127 
128         if(j!=-1):
129             for k in range(j+9,len(m[i])):
130                 if(m[i][k]==')'):
131                     n=k+2
132                     break
133 
134         if(j==-1 or m[i][j+9]==')'):
135             content=content.replace(m[i][8:n-1],m[i][n:])
136             continue
137         else:
138             y_param=m[i][j+9:n-2].split(',')
139 
140             pattern=m[i][8:j+8]+'\(.*?\)'
141             m2=re.findall(pattern,content)
142 
143             for z in m2:
144                 param=z[j+1:-1].split(',')
145                 st=m[i][n:]
146 
147 
148                 for k in range(len(param)):
149                     st=re.sub(y_param[k].strip(),param[k],st)
150                 content=content.replace(z,st)
151     return content
152 
```

3.1.1.6 remove_redundant_functions()

```
def models.remove_redundant_functions (
               content )
```

```
Arguments:
    content: string storing the source code
Return type: updated string
Functionality:
    redundant functions in a source code are removed
Logic Used:
     Write a minimal parser that can identify functions.
     It just needs to detect the start and ending line of a function.
     Programmatically comment out the first function, save to a temp file.
     Try to compile the file by invoking the complier.
     Check if there are compile errors, if yes, the function is called, if not, it is unused.
     Continue with the next function.
     Reference:https://stackoverflow.com/questions/33209302/removal-of-unused-or-redundant-code
```

```
23 def remove_redundant_functions(content):
24     """
25     Arguments:
26         content: string storing the source code
27     Return type: updated string
28     Functionality:
29         redundant functions in a source code are removed
30     Logic Used:
31          Write a minimal parser that can identify functions.
32          It just needs to detect the start and ending line of a function.
33          Programmatically comment out the first function, save to a temp file.
34          Try to compile the file by invoking the complier.
35          Check if there are compile errors, if yes, the function is called, if not, it is unused.
36          Continue with the next function.
37          Reference:https://stackoverflow.com/questions/33209302/removal-of-unused-or-redundant-code
38     """
39     type_list=['int','void','char','string']
40     t=""
41     for type in type_list:
42         L=re.findall(type+"\s*[a-z0-9_]\s\([a-z0-9_ \n\t,\r\f\v]\)\s\{",content)
43         for y in L:
44             y=y.replace("(","\(")
45             y=y.replace(")","\)")
46             x=re.search(y,content)
47             first=x.span()[0]
```

```
48             last=x.span()[1]
49             count=1
50             while(count!=0):
51                 if(content[last]=='{'):
52                     count+=1
53                 if(content[last]=='}'):
54                     count-=1
55                 last+=1
56             t=content[0:first]+content[last:]
57         temp = open("temp.cpp", "w")
58         temp.write(t)
59         temp.close()
60         g = subprocess.getstatusoutput("g++ temp.cpp")
61         if(g[1]==""):
62             content=t
63     return content
64
```

3.1.1.7 similarity()

```
def models.similarity (
              s,
              t )
```

```
Arguments    :
    s   :(sorted)list of numbers
    t   :(sorted)list of numbers
Return type  :
    returns a number in the range(0,1)
Functionality:
    Evaluates the cosine product of the two vectors
```

```
325 def similarity(s,t):
326
327     """
328     Arguments    :
329         s   :(sorted)list of numbers
330         t   :(sorted)list of numbers
331     Return type  :
332         returns a number in the range(0,1)
333     Functionality:
334         Evaluates the cosine product of the two vectors
335     """
336     x=min(s.size,t.size)
337     s=s[-x:]
338     t=t[-x:]
339     return np.dot(s,t)/(np.linalg.norm(s)*np.linalg.norm(t))
```

3.1.1.8 tf_idf()

```
def models.tf_idf (
              word_count_in_each_file,
              word_count_across_documents,
              list_of_paths,
              list_of_files )
```

```
 Arguments    :
    list_of_paths              :list of source code files
    list_of_files                  :It consists data of all the Users who have been SignedUp
    word_count_in_each_file    :Frequency of word corresponding to each file as an array of dictionary
    word_count_across_documents:Frequency of each word across as files corresponding to a particular assignmen
Functionality:
    It computes tf_idf vector corresponding to each file.
```

```
    The tf_idf function is somewhat different from the original one
    If we use the bag of words strategy then similarity is determined mostly by the variables which have maxim
    But similarity should depend more on core logic loke number of functions,operators loops etc.
    The weight added for each word say 'x' in file 'f' is log(freq of x across all files corresponding to assi
    Words which have low frequency in a file than average frequency across all files are given +ve weightage
    Words which have high frequency in a file than average frequency across all files are given -ve weightage
    Uniqueness is determined by high weightage words.

255 def tf_idf(word_count_in_each_file,word_count_across_documents,list_of_paths,list_of_files):
256     """ Arguments    :
257         list_of_paths              :list of source code files
258         list_of_files                 :It consists data of all the Users who have been SignedUp
259         word_count_in_each_file    :Frequency of word corresponding to each file as an array of
      dictionary
260         word_count_across_documents:Frequency of each word across as files corresponding to a particular
      assignment as dictionary
261      Functionality:
262         It computes tf_idf vector corresponding to each file.
263         The tf_idf function is somewhat different from the original one
264         If we use the bag of words strategy then similarity is determined mostly by the variables which
      have maximum count in a file.
265         But similarity should depend more on core logic loke number of functions,operators loops etc.
266         The weight added for each word say 'x' in file 'f' is log(freq of x across all files
      corresponding to assignment/(freq of x in f*number of files))
267         Words which have low frequency in a file than average frequency across all files are given +ve
      weightage
268         Words which have high frequency in a file than average frequency across all files are given -ve
      weightage
269         Uniqueness is determined by high weightage words.
270      """
271     similarity_matrix=np.zeros((len(list_of_paths),len(list_of_paths)))
272     tf_idf_vec=[]
273     for i in range(len(list_of_paths)):
274         temp=[]
275         for j in word_count_in_each_file[i]:
276
      temp.append(word_count_in_each_file[i].get(j)*(-0.01+(math.log(word_count_across_documents.get(j)/word_count_in_each_f
277         temp.sort()
278         tf_idf_vec.append(temp)
279
280     for i in range(len(list_of_paths)):
281         similarity_matrix[i,i]=1;
282         for j in range(i+1,len(list_of_paths)):
283             similarity_matrix[i,j]=similarity(np.array(tf_idf_vec[i]),np.array(tf_idf_vec[j]))
284             similarity_matrix[j,i]=similarity_matrix[i,j]
285
286     txt_file(similarity_matrix,list_of_paths,list_of_files)
287
```

3.1.1.9 txt_file()

```
def models.txt_file (
              similarity_matrix,
              list_of_paths,
              list_of_files )


Arguments    :
    list_of_paths    :list of source code file names
    similarity_matrix:2-dimensional matrix representing mutual similarity between each pair of files
    list_of_files        :It consists data of all the Users who have been SignedUp
Functionality:
    Displaying the Percentage matching among files in text format and saving it as a csv file

288 def txt_file(similarity_matrix,list_of_paths,list_of_files):
289     """
290     Arguments    :
291         list_of_paths    :list of source code file names
292         similarity_matrix:2-dimensional matrix representing mutual similarity between each pair of files
293         list_of_files        :It consists data of all the Users who have been SignedUp
294     Functionality:
295         Displaying the Percentage matching among files in text format and saving it as a csv file """
296     result=open("media/result.txt","w")
297     res=""
298     for i in range(len(list_of_paths)):
```

```
299         for j in range(i+1,len(list_of_paths)):
300             res+="similarity between "+ list_of_files[i]+list_of_files[j]+" =
      "+str(similarity_matrix[i][j])+"\n"
301      result.write(res)
302      csv_file(list_of_files,similarity_matrix)
303
```

3.1.1.10 visualizer()

```
def models.visualizer (
              list_of_files,
              similarity_matrix )
```

```
Arguments     :
    list_of_files    :list of source code file names
    similarity_matrix:2-dimensional matrix representing mutual similarity between each pair of files
Return type   :
    Path of the saved image
Functionality:
    Plotting the output similarity_matrix and saving it as an image
```

```
65 def visualizer(list_of_files,similarity_matrix):
66     """
67     Arguments     :
68         list_of_files    :list of source code file names
69         similarity_matrix:2-dimensional matrix representing mutual similarity between each pair of files
70     Return type   :
71         Path of the saved image
72     Functionality:
73         Plotting the output similarity_matrix and saving it as an image """
74     x=range(len(list_of_files))
75     y=range(len(list_of_files))
76     xx,yy=np.meshgrid(x,y)
77     z=similarity_matrix[xx,yy]
78     cmap =cm.get_cmap("rainbow",100)
79     fig=plt.figure()
80     ax=fig.add_subplot(111)
81     im=ax.matshow(z,cmap=cmap,vmin=0,vmax=1.01,origin='lower')
82     for i in range(len(list_of_files)):
83         ax.text(i,i,"none",ha="center", va="center", color="k")
84         for j in range(i+1,len(list_of_files)):
85             ax.text(j, i, int(similarity_matrix[i,j]*100)/100,ha="center", va="center", color="k")
86             ax.text(i,j, int(similarity_matrix[i,j]*100)/100,ha="center", va="center", color="k")
87     fig.colorbar(im,shrink=0.5)
88     ax.set_xticks(range(len(list_of_files)))
89     ax.set_yticks(range(len(list_of_files)))
90     ax.set_xticklabels(list_of_files,rotation=90)
91     ax.set_yticklabels(list_of_files)
92     ax.tick_params(labelsize=30/len(list_of_files))
93     random=np.random.randint(1,100)
94     path='result.png'
95     plt.savefig('media/'+path)
96
```

3.2 views Namespace Reference

Classes

- class DataUpload
- class UserViewSet

Functions

- def create_user (request)

3.2.1 Function Documentation

3.2.1.1 create_user()

```
def views.create_user (
                request )
```

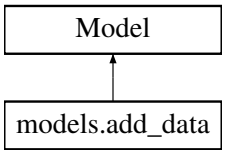
```
21 def create_user(request):
22     validated_data=JSONParser().parse(request)
23     if(validated_data['password']!=validated_data['password2']):
24         return JsonResponse("Passwords donot match",safe=False)
25     elif(User.objects.filter(email=validated_data['email']).exists()):
26         return JsonResponse("There exists an account with this email",safe=False)
27     elif(User.objects.filter(username=validated_data['username']).exists()):
28         return JsonResponse("There exists an account with this username",safe=False)
29     else:
30         try:
31             new_data={}
32             new_data['username']=validated_data['username']
33             new_data['password']=validated_data['password']
34             new_data['email']=validated_data['email']
35         except:
36             return JsonResponse("failed",safe=False)
37         user_serializer=UserSerializer(data=new_data)
38         if user_serializer.is_valid():
39             user_serializer.save()
40             return JsonResponse("success",safe=False)
41         return JsonResponse("Invalid credentials",safe=False)
42
```

Chapter 4

Class Documentation

4.1 models.add_data Class Reference

Inheritance diagram for models.add_data:



Static Public Attributes

- username = models.CharField(max_length=50)
- label = models.CharField(max_length=50)
- data = models.FileField()

4.1.1 Detailed Description

4.1.2 Member Data Documentation

4.1.2.1 data

```
models.add_data.data = models.FileField()  [static]
```

4.1.2.2 label

```
models.add_data.label = models.CharField(max_length=50)  [static]
```

4.1.2.3 username

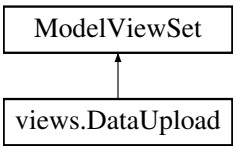
```
models.add_data.username = models.CharField(max_length=50)  [static]
```

The documentation for this class was generated from the following file:

- models.py

4.2 views.DataUpload Class Reference

Inheritance diagram for views.DataUpload:



Public Member Functions

- def post (request, *args, **kwargs)
- def view_files (request)

Static Public Attributes

- tuple authentication_classes = (TokenAuthentication,)
- tuple permission_classes = (IsAuthenticated,)
- tuple parser_classes = (MultiPartParser, FormParser)
- serializer_class = DataSerializer

4.2.1 Member Function Documentation

4.2.1.1 post()

```
def views.DataUpload.post (
                request,
               * args,
               ** kwargs )
```

```
49     def post(request, *args, **kwargs):
50         label=request.POST['label']
51         username=request.POST['username']
52         your_file = request.FILES['data']
53         add_data.objects.create( username=username,label=label,data=your_file)
54         t=tarfile.open('media/'+str(your_file),'r')
55         L=t.getnames()
56         t.extractall(path='media/')
57         return JsonResponse("success",safe=False)
58
```

4.2.1.2 view_files()

```
def views.DataUpload.view_files (
                request )
```

```
60     def view_files(request):
61         data = request.POST
62         your_files=add_data.objects.filter(label=data['label'])
63         if(not your_files.exists()):
64             return JsonResponse("You dont have any files",safe=False)
65         l=[]
66         users=[]
67         for i in your_files:
68             t=tarfile.open('media/'+str(i.data.name),'r')
69             L=t.getnames()
70             for k in L:
71                 l.append(k)
72                 try:
73                     users.append(k.split('/')[1])
74                 except:
75                     pass
76         preprocessing(l,users)
77         data={'png':'result.png','txt':'result.txt','csv':'result.csv'}
78         return JsonResponse(data, safe=False)
```

4.2.2 Member Data Documentation

4.2.2.1 authentication_classes

```
tuple views.DataUpload.authentication_classes = (TokenAuthentication,)  [static]
```

4.2.2.2 parser_classes

```
tuple views.DataUpload.parser_classes = (MultiPartParser, FormParser)  [static]
```

4.2.2.3 permission_classes

```
tuple views.DataUpload.permission_classes = (IsAuthenticated,)  [static]
```

4.2.2.4 serializer_class

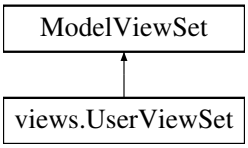
```
views.DataUpload.serializer_class = DataSerializer  [static]
```

The documentation for this class was generated from the following file:

- views.py

4.3 views.UserViewSet Class Reference

Inheritance diagram for views.UserViewSet:



Static Public Attributes

- queryset = User.objects.all().order_by('username')
- serializer_class = UserSerializer
- tuple authentication_classes = (TokenAuthentication,)
- list permission_classes = [IsAuthenticated]

4.3.1 Member Data Documentation

4.3.1.1 authentication_classes

```
tuple views.UserViewSet.authentication_classes = (TokenAuthentication,)  [static]
```

4.3.1.2 permission_classes

```
list views.UserViewSet.permission_classes = [IsAuthenticated]  [static]
```

4.3.1.3 queryset

```
views.UserViewSet.queryset = User.objects.all().order_by('username')  [static]
```

4.3.1.4 serializer_class

```
views.UserViewSet.serializer_class = UserSerializer  [static]
```

The documentation for this class was generated from the following file:

- views.py

Index