## Code Frisk

Generated by Doxygen 1.8.20

1 Hierarchical Index		1
1.1 Class Hierarchy		1
2 Class Index		3
2.1 Class List		3
3 Namespace Documentation		5
3.1 models Namespace Reference		5
3.1.1 Function Documentation		5
3.1.1.1 csv_file()		5
3.1.1.2 preprocessing()		6
3.1.1.3 remove_comments()		7
3.1.1.4 remove_comments_pythonfile()		7
3.1.1.5 remove_macros()		8
3.1.1.6 remove_redundant_functions()		9
3.1.1.7 similarity()		10
3.1.1.8 tf_idf()		10
3.1.1.9 txt_file()		11
3.1.1.10 visualizer()		12
3.2 views Namespace Reference		12
3.2.1 Function Documentation		12
3.2.1.1 create_user()		13
4 Class Documentation		15
4.1 models.add_data Class Reference		15
4.1.1 Detailed Description		15
4.1.2 Member Data Documentation		15
4.1.2.1 data		15
4.1.2.2 label		16
4.1.2.3 username		16
4.2 views.DataUpload Class Reference		16
4.2.1 Member Function Documentation		16
4.2.1.1 post()		17
4.2.1.2 view_files()		17
4.2.2 Member Data Documentation		17
4.2.2.1 authentication_classes		17
4.2.2.2 parser_classes		18
4.2.2.3 permission_classes		18
4.2.2.4 serializer_class		18
4.3 views.UserViewSet Class Reference		18
4.3.1 Member Data Documentation		18
4.3.1.1 authentication_classes		18
4.3.1.2 permission_classes		19
_		_

Index		21
	4.3.1.4 serializer_class	19
	4.3.1.3 queryset	19

# **Chapter 1**

# **Hierarchical Index**

## 1.1 Class Hierarchy

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

Model	
models.add_data	15
ModelViewSet	
views.DataUpload	16
views.UserViewSet	18

2 Hierarchical Index

# Chapter 2

# **Class Index**

### 2.1 Class List

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

nodels.add_data	15
riews.DataUpload	16
views.UserViewSet	18

4 Class Index

## **Chapter 3**

# **Namespace Documentation**

### 3.1 models Namespace Reference

### **Classes**

· class add\_data

### **Functions**

- def remove\_redundant\_functions (content)
- def remove\_comments (string)
- def visualizer (list\_of\_files, similarity\_matrix)
- def remove\_macros (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()

```
300 def csv_file(list_of_files, similarity_matrix):
        #""" Interpreting the Output data as a CSV file ,
        #where each element represent the percentage matching between the file
#corresponding to a row and column"""
302
303
304
305
        Arguments
            list_of_files
306
                             :list of source code file names
307
            similarity_matrix:2-dimensional matrix representing mutual similarity between each pair of files
308
        Functionality:
309
            Plotting the output similarity_matrix and saving it as an csv file
310
311
        f=similarity_matrix.tolist()
312
313
        files=['filenames']+list_of_files
314
        for x in range(len(list_of_files)):
315
            f[x] = [list_of_files[x]] + f[x]
        f=[files]+f
316
        with open("media/result.csv", "w+") as myCsv:
317
            csvWriter = csv.writer(myCsv, delimiter=',')
318
            csvWriter.writerows(f)
        visualizer(list_of_files, similarity_matrix)
320
```

#### 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 source code paths
    list_of_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_
171 def preprocessing(list_of_paths,list_of_files):
172
173
174
       Core logic is based on the Bag_of_Words
       and TF-IDF Strategy( Term frequency and Inverse Document Frequency )
175
176
       Arguments
                          :list of source code paths
177
           list_of_paths
178
          list_of_files
                          :it consists of list of file names
179
       Functionality:
180
           It finds the count of each word after removing comments and replacing macros and passes this
      vector to tf_idf function.
181
182
183
       word_count_across_documents={ }
184
       """ Maintains the word count of each element in a document """
185
186
       word count in each file=[]
187
188
       for files in list_of_paths:
189
190
           filename='media/'+files
           temp={}
191
           myfile=open(filename, "r")
192
193
           content=myfile.read()
194
          myfile.close()
195
           if(files[-4:]=='.cpp'):
196
197
              content=remove_redundant_functions(content)
198
              content=remove_macros(content)
199
              content=remove comments(content)
           if(files[-5:]=='.java'):
200
201
              content=remove_comments(content)
202
           elif(files[-3:]=='.py'):
203
              content=remove_comments_pythonfile(content)
2.04
      205
           for i in sym:
              content=content.replace(i," "+i+" ")
206
```

```
207
                     if(files[-4:]=='.cpp' or files[-5:]=='.java'):
    content=content.replace("while","for")
    content=content.replace("switch","if")
208
209
210
                            content=content.replace("case", "else if")
content=content.replace("default", "else")
content=content.replace("unsigned long long int", "double")
211
212
213
                           content=content.replace("unsigned long long", "double")
content=content.replace("long long int", "double")
content=content.replace("long long", "double")
content=content.replace("float", "double")
content=content.replace("float", "double")
214
215
216
217
218
                           content=content.replace("int", "double")
content=content.replace("for", "double")
content=content.replace("+ +","+ = 1")
content=content.replace("- -","- = 1")
content=content.replace("< <","«")
content=content.replace('> >',"»")
219
220
221
222
223
                            cont=""
224
                            for i in content.split('\n'):
225
                                   if(i==' \setminus n'):
226
227
228
                                   i=i.strip()
                                   if (len(i) == 0 \text{ or } i[0] == ' #'):
229
230
                                          continue
231
                                   else:
                                          cont=cont+' '+i
232
233
                            content=cont
234
                    elif(files[-3:]=='.py'):
235
                            content=content.replace('while','for')
236
                            content=content.replace('switch','if')
237
                            content=content.replace('case','elif')
content=content.replace('default','else')
238
239
                    content=content.replace('do',")
content=re.sub(':|\'|\"',",content)
List_of_words=content.split()
240
241
242
243
244
                     for i in List_of_words:
245
                            temp[i] = temp.get(i, 0) + 1
246
                            word_count_across_documents[i]=word_count_across_documents.get(i,0)+1
247
2.48
                     word_count_in_each_file.append(temp)
249
              tf_idf(word_count_in_each_file,word_count_across_documents,list_of_paths,list_of_files)
250
```

### 3.1.1.3 remove\_comments()

```
def models.remove_comments (
                                                                         string )
 66 def remove_comments(string):
                                pattern = r''(\ ... ?\ '') | (/\ ... ?\ '/) | (/\ ... ?\ '/) | (/\ ... ?\ ') | (/\ ... ?\ ') | (/\ ... ?\ ') | (/\ ... ?\ ') | (/\ ... ?\ ') | (/\ ... ?\ ') | (/\ ... ?\ ') | (/\ ... ?\ ') | (/\ ... ?\ ') | (/\ ... ?\ ') | (/\ ... ?\ ') | (/\ ... ?\ ') | (/\ ... ?\ ') | (/\ ... ?\ ') | (/\ ... ?\ ') | (/\ ... ?\ ') | (/\ ... ?\ ') | (/\ ... ?\ ') | (/\ ... ?\ ') | (/\ ... ?\ ') | (/\ ... ?\ ... ?\ ') | (/\ ... ?\ ... ?\ ... | (/\ ... ?\ ... ?\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ ... |\ 
                                  # first group captures quoted strings (double or single)
68
                                 \# second group captures comments (//single-line or /* multi-line */)
69
                                regex = re.compile(pattern, re.MULTILINE|re.DOTALL)
70
71
                                 def _replacer(match):
 72
                                                 # if the 2nd group (capturing comments) is not None,
                                                     # it means we have captured a non-quoted (real) comment string.
                                                   if match.group(2) is not None:
    return "" # so we will return empty to remove the comment
else: # otherwise, we will return the 1st group
    return match.group(1) # captured quoted-string
74
7.5
76
77
78
                                 return regex.sub(_replacer, string)
```

### 3.1.1.4 remove comments pythonfile()

```
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 '''
156 def remove_comments_pythonfile(file_content):
157
158
        Arguments:
            file_content: string storing the source code in python
159
160
        Return type: updated string
161
        Functionality:
162
            All commments in the code are replaced.
163
        Logic Used:
            Using Regex detect substrings starting with \# and ending with \ Similarly detect substrings starting with \# and ending with \#
164
165
166
167
        pattern=re.compile('#.*?$|\'\'.*?\'\'\"\"\".*?\"\"\",re.DOTALL|re.MULTILINE)
168
        content=re.sub(pattern,",file_content)
169
        return content
170
```

### 3.1.1.5 remove\_macros()

```
def models.remove_macros (
                 content )
Arguments:
    content: string storing the source code
Return type: updated string
Functionality:
    All macros in the code are replaced.
Logic Used:
     This is based on preprocessing done by g++ compiler
This function assumes the existence of "using namespace std;" as a substring in the source code string
     g++ -E file.cpp produces the preprocessed code which does not contain any comments or macro
     The substring following "using namespace std;" is extracted
     Using Regex to detect the typedef macros
     Replace them using replace() function
111 def remove_macros(content):
112
113
114
115
        Arguments:
116
            content: string storing the source code
         Return type: updated string
117
118
        Functionality:
119
            All macros in the code are replaced.
        Logic Used:
120
             This is based on preprocessing done by g++ compiler
This function assumes the existence of "using namespace std;" as a substring in the source code
121
122
        string
             g++ -E file.cpp produces the preprocessed code which does not contain any comments or macro The substring following "using namespace std;" is extracted
123
124
             Using Regex to detect the typedef macros
125
        Replace them using replace() function
126
127
128
         temp = open("temp.cpp", "w")
129
         temp.write(content)
130
         temp.close()
         pre = subprocess.getstatusoutput("g++ -E temp.cpp")
131
        prep=pre[1].split('using namespace std;')[-1]
132
133
         content=prep
134
        m=re.findall('typedef .+ .+',content)
"""finding macros"""
135
136
137
        content=re.sub('typedef .+ .+',",content)
"""removing macros definitions"""
138
139
140
```

```
for i in range(len(m)):
    """replacing macros"""
141
142
143
             j=m[i].split()
             if(j[-1] ==';'):
144
145
                  last_word=j[-2]
146
             else:
147
                  last_word=j[-1]
148
                  if(last_word[-1]==';'):
                  last_word=last_word[:-1]
string=""
149
150
                  for i in range(1,len(j)-1):
151
                      string+=j[i]+' '
152
                  content=content.replace(last_word, string)
153
154
155
```

#### 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.
     {\tt Reference:https://stackoverflow.com/questions/33209302/removal-of-unused-or-redundant-code}
24 def remove_redundant_functions(content):
25
26
       Arguments:
27
           content: string storing the source code
2.8
       Return type: updated string
29
       {\tt Functionality:}
          redundant functions in a source code are removed
30
31
       Logic Used:
            Write a minimal parser that can identify functions.
33
            It just needs to detect the start and ending line of a function.
34
            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.
35
36
            Continue with the next function.
Reference:https://stackoverflow.com/questions/33209302/removal-of-unused-or-redundant-code
37
38
39
       type_list=['int','void','char','string','double']
40
41
42
       for type in type_list:
           43
44
           for y in L:
               y=y.replace("(","\(")
y=y.replace(")","\)")
45
46
47
               x=re.search(y,content)
48
               first=x.span()[0]
49
               last=x.span()[1]
50
               count=1
               while (count!=0):
52
                 if(content[last] == '{'):
53
                       count+=1
                   if(content[last] == ' }'):
54
55
                      count-=1
                   last+=1
56
               t=content[0:first]+content[last:]
          temp = open("temp.cpp", "w")
59
          temp.write(t)
60
           temp.close()
           g = subprocess.getstatusoutput("g++ temp.cpp")
61
          if(g[1]==""):
62
63
               content=t
       return content
65
```

### 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 % \left( \mathbf{r}\right) =\left( \mathbf{r}\right) 
321 def similarity(s,t):
322
323
324
        Arguments
            s :(sorted)list of numbers
t :(sorted)list of numbers
325
326
327
        Return type :
328
            returns a number in the range (0,1)
        Functionality:
329
        Evaluates the cosine product of the two vectors
330
331
332
        x=np.zeros(abs(s.size-t.size))
""
333
334
        if(s.size>t.size):
335
           t=np.concatenate((x,t))
        else:
336
        s=np.concatenate((x,s))
337
338
        x=min(s.size,t.size)
339
340
         s=s[-x:]
341
        t=t[-x:]
342
343
        \#s=(s-np.mean(s))/np.std(s)
344
        #t = (t-np.mean(t))/np.std(t)
345
        return 1-np.linalg.norm(s-t)/(np.linalg.norm(s)+np.linalg.norm(t))
346
        return np.dot(s,t)/(np.linalg.norm(s)*np.linalg.norm(t))
```

Uniqueness is determined by high weightage words.

### 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 assignment
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 assistance).
```

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

```
251 def tf_idf(word_count_in_each_file,word_count_across_documents,list_of_paths,list_of_files):
         """ Arguments
252
253
             list_of_paths
                                           :list of source code files
                                               :It consists data of all the Users who have been {\tt SignedUp}
254
             list_of_files
             word_count_in_each_file
255
                                         :Frequency of word corresponding to each file as an array of
       dictionary
256
             word_count_across_documents:Frequency of each word across as files corresponding to a particular
       assignment as dictionary
257
        Functionality:
2.58
             It computes tf_idf vector corresponding to each file.
             The tf\_idf function is somewhat different from the original one
259
             If we use the bag of words strategy then similarity is determined mostly by the variables which
260
       have maximum count in a file.
             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
261
262
       corresponding to assignment/(freq of x in f*number of files))
263
             Words which have low frequency in a file than average frequency across all files are given +ve
       weightage
264
             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. \ensuremath{\text{\mbox{\tt min}}}
265
266
        \verb|similarity_matrix=np.zeros((len(list_of_paths),len(list_of_paths)))|\\
2.67
268
         tf_idf_vec=[]
269
         for i in range(len(list_of_paths)):
270
             temp=[]
             for j in word_count_in_each_file[i]:
271
272
       \texttt{temp.append(word\_count\_in\_each\_file[i].get(j)*((math.log(word\_count\_across\_documents.get(j)/word\_count\_in\_each\_file[i]))} \\
273
             temp.sort()
274
             tf idf vec.append(temp)
275
276
         for i in range(len(list_of_paths)):
277
             similarity_matrix[i,i]=0
278
             for j in range(i+1,len(list_of_paths)):
                  similarity_matrix[i,j]=similarity(np.array(tf_idf_vec[i]),np.array(tf_idf_vec[j]))
279
280
                  similarity_matrix[j,i]=similarity_matrix[i,j]
281
282
        txt_file(similarity_matrix,list_of_paths,list_of_files)
283
```

### 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
284 def txt_file(similarity_matrix,list_of_paths,list_of_files):
285
286
       Arguments
287
            list of paths
                            :list of source code file names
            similarity_matrix:2-dimensional matrix representing mutual similarity between each pair of files
288
            list_of_files
289
                                :It consists data of all the Users who have been SignedUp
290
       Functionality:
291
           Displaying the Percentage matching among files in text format and saving it as a csv file """
292
       result=open("media/result.txt","w")
       res=""
293
294
       for i in range(len(list_of_paths)):
295
           for j in range(i+1,len(list_of_paths)):
       res+="similarity between "+ list_of_files[i]+" and "+list_of_files[j]+" = "+str(similarity_matrix[i][j])+"\n"
296
297
       result.write(res)
       csv_file(list_of_files, similarity_matrix)
298
299
```

### 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
80 def visualizer(list_of_files,similarity_matrix):
       Arguments
82
          list_of_files
                            :list of source code file names
83
           similarity_matrix:2-dimensional matrix representing mutual similarity between each pair of files
84
       Return type
          Path of the saved image
87
     Functionality:
          Plotting the output similarity_matrix and saving it as an image """
88
      #x=range(len(list_of_files))
#y=range(len(list_of_files))
89
90
       \#xx,yy=np.meshgrid(x,y)
       #z=similarity_matrix[xx,yy]
93
       \#z=np.round(z*100)
94
       \verb|cmap = matplotlib.colors.LinearSegmentedColormap.from_list("", ["green", "yellow", "red"])| \\
9.5
       fig=plt.figure()
       ax=fig.add_subplot(111)
96
       im=ax.matshow(similarity_matrix, cmap=cmap, vmin=0, vmax=1, origin='lower')
98
       for i in range(len(list_of_files)):
99
          for j in range(i+1,len(list_of_files)):
       ax.text(j, i, int(similarity_matrix[i,j]*100),ha="center", va="center",
color="k",fontsize=50/(len(list_of_files)+1))
100
                ax.text(i,j, int(similarity_matrix[i,j]*100),ha="center", va="center",
101
       color="k", fontsize=50/(len(list_of_files)+1))
102
       fig.colorbar(im, shrink=0.5)
103
        ax.set_xticks(range(len(list_of_files)))
104
        ax.set_yticks(range(len(list_of_files)))
105
        ax.set_xticklabels(list_of_files,rotation=90)
        ax.set_yticklabels(list_of_files)
106
107
       random=np.random.randint(1,100)
        path='result.png'
109
       plt.savefig('media/'+path)
110
```

### 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()

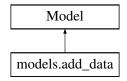
```
def views.create_user (
20 def create_user(request):
21    validated_data=JSONParser().parse(request)
22    if(validated_data['password']!=validated_data['password2']):
                  return JsonResponse("Passwords donot match", safe=False)
           elif(User.objects.filter(email=validated_data['email']).exists()):
           return JsonResponse("There exists an account with this email",safe=False)
elif(User.objects.filter(username=validated_data['username']).exists()):
    return JsonResponse("There exists an account with this username",safe=False)
25
26
2.7
           else:
28
                 try:
30
                        new_data={}
                        new_data('username')=validated_data('username')
new_data('password')=validated_data('password')
new_data('email')=validated_data('email')
31
32
33
34
                 except:
                       return JsonResponse("failed", safe=False)
35
                 user_serializer=UserSerializer(data=new_data)
37
                 if user_serializer.is_valid():
38
                        user_serializer.save()
                 return JsonResponse("success", safe=False)
return JsonResponse("Invalid credentials", safe=False)
39
40
```

## **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

16 Class Documentation

### 4.1.2.2 label

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

### 4.1.2.3 username

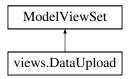
```
models.add_data.username = models.CharField(max_length=50) [static]
```

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 )
        def post(request, *args, **kwargs):
    label=request.POST['label']
48
49
             username=request.POST['username']
50
             your_file = request.FILES['data']
             add_data.objects.create( username=username, label=label, data=your_file)
             if(your_file.name.find('.tar')!=-1):
    t=tarfile.open('media/'+str(your_file),'r')
53
54
55
                  L=t.getnames()
                 t.extractall(path='media/')
56
             else:
58
59
             return JsonResponse("success", safe=False)
60
```

### 4.2.1.2 view\_files()

```
def views.DataUpload.view_files (
                request )
       def view_files(request):
63
           data = request.POST
           your_files=add_data.objects.filter(label=data['label'])
64
65
           if(not your_files.exists()):
               return JsonResponse("You dont have any files", safe=False)
66
68
           h=[]
69
           for i in your_files:
               if(i.data.name.find('.tar')!=-1):
70
                    t=tarfile.open('media/'+str(i.data.name),'r')
71
72
                    L=t.getnames()
73
                    for k in L:
74
                        try:
75
                            if(k.find('._')!=-1 or k.find('.DS_Store')!=-1):
76
                            \texttt{h.append(k.split('/')[1])}
77
78
                            1.append(k)
79
                        except:
81
                    1.append(i.data.name)
82
8.3
                   h.append(i.data.name)
           preprocessing(l,h)
data={'png':'result.png','txt':'result.txt','csv':'result.csv'}
84
85
           return JsonResponse (data, safe=False)
```

### 4.2.2 Member Data Documentation

### 4.2.2.1 authentication\_classes

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

18 Class Documentation

### 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

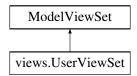
```
views.DataUpload.serializer_class = DataSerializer [static]
```

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

20 Class Documentation

# Index

authentication_classes	models, 9
views.DataUpload, 17	
views.UserViewSet, 18	serializer_class
	views.DataUpload, 18
create_user	views.UserViewSet, 19
views, 12	similarity
csv_file	models, 9
models, 5	tf idf
data	models, 10
models.add data, 15	txt file
models.add_data, 13	models, 11
label	models, TT
models.add_data, 15	username
,	models.add_data, 16
models, 5	_ ,
csv_file, 5	view_files
preprocessing, 6	views.DataUpload, 17
remove_comments, 7	views, 12
remove_comments_pythonfile, 7	create_user, 12
remove_macros, 8	views.DataUpload, 16
remove_redundant_functions, 9	authentication_classes, 17
similarity, 9	parser_classes, 17
tf_idf, 10	permission_classes, 18
txt_file, 11	post, 16
visualizer, 11	serializer_class, 18
models.add_data, 15	view_files, 17
data, 15	views.UserViewSet, 18
label, 15	authentication_classes, 18
username, 16	permission_classes, 18
	queryset, 19
parser_classes	serializer_class, 19
views.DataUpload, 17	visualizer
permission_classes	models, 11
views.DataUpload, 18	
views.UserViewSet, 18	
post	
views.DataUpload, 16	
preprocessing models, 6	
models, 6	
queryset	
views.UserViewSet, 19	
The state of the s	
remove_comments	
models, 7	
remove_comments_pythonfile	
models, 7	
remove_macros	
models, 8	
remove_redundant_functions	