SENDGRID INTEGRATION WITH PYTHON

Date	15 Nov 2022
Team ID	PNT2022TMID08156
Project Name	SKILL/JOB RECOMMENDED APP

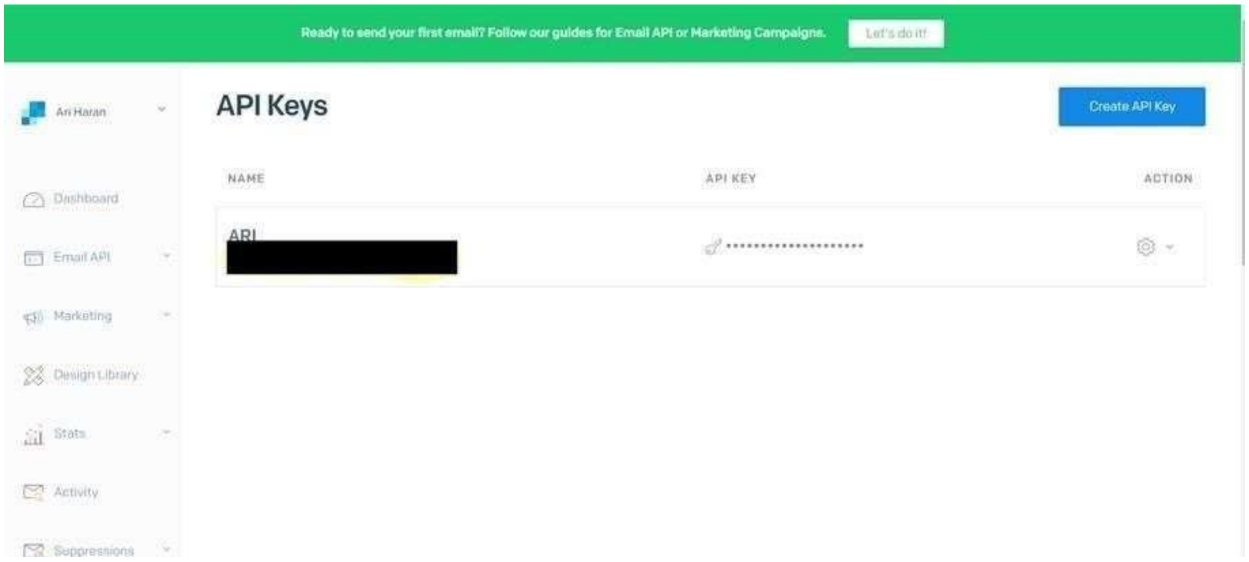
STEP 1:

Requirements:

Python 2.6, 2.7, 3.4 or 3.5.

STEP 2:

Creating an API key

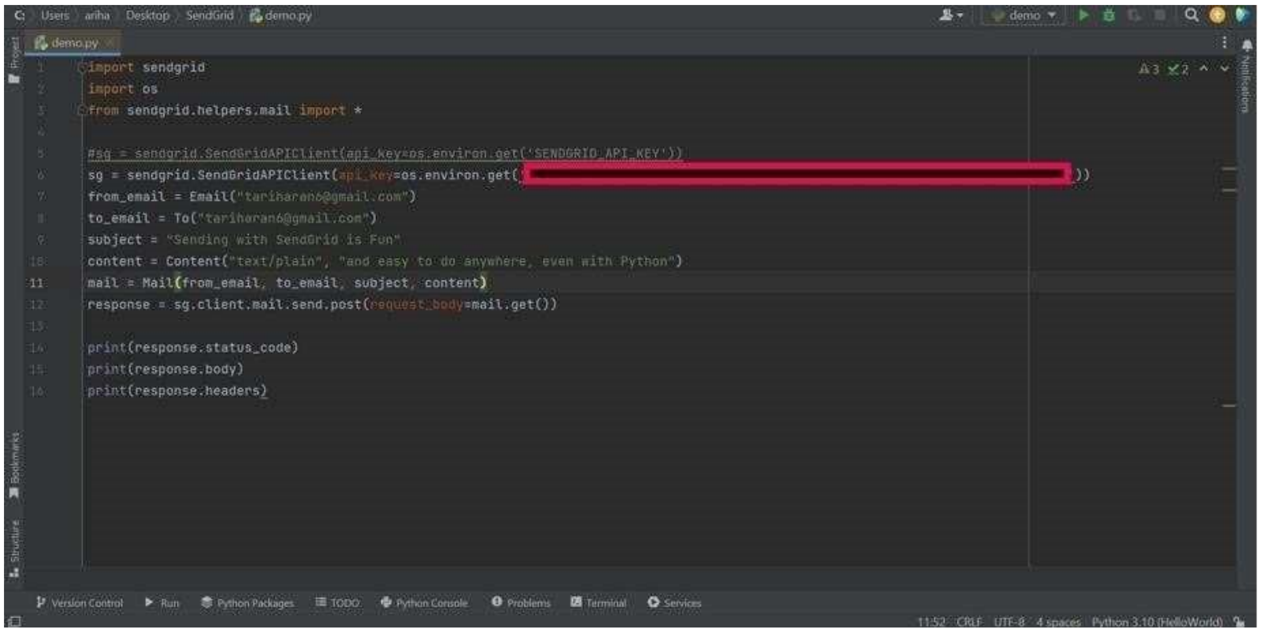


[redacted]

STEP 3: INSTALL PAKAGE: > pip install sendgrid

SETP 4:

SEND EMAIL



```python
import sendgrid
import os
from sendgrid.helpers.mail import *

#sg = sendgrid.SendGridAPIClient(api_key=os.environ.get('SENDGRID_API_KEY'))
sg = sendgrid.SendGridAPIClient(api_key=os.environ.get([redacted]))
from_email = Email("tariharan6@gmail.com")
to_email = To("tariharan6@gmail.com")
subject = "Sending with SendGrid is Fun"
content = Content("text/plain", "and easy to do anywhere, even with Python")
mail = Mail(from_email, to_email, subject, content)
response = sg.client.mail.send.post(request_body=mail.get())

print(response.status_code)
print(response.body)
print(response.headers)
```

SENDGRID PYTHON CODE :

```python
"""HTTP Client library"""
import json
import logging
from .exceptions import handle_error

    try:
    # Python 3
    import urllib.request as urllib
    from urllib.parse import urlencode
    from urllib.error import HTTPError
    except ImportError:
    # Python 2
```

```python
import os
```

```
from sendgrid import SendGridAPIClient
from sendgrid.helpers.mail import Mail

    message = Mail(
    from_email='from_email@example.com',
    to_emails='to@example.com',
    subject='Sending with Twilio SendGrid is Fun',
    html_content='<strong>and easy to do anywhere, even with
    Python</strong>')
    try:
    sg = SendGridAPIClient(os.environ.get('SENDGRID_API_KEY'))
    response = sg.send(message)
    print(response.status_code)
    print(response.body) print(response.headers) except Exception as
    e:

    print(e.message)
```

HTTP CLIENT PROGRAM:

```
    import urllib2 as urllib
    from urllib2 import HTTPError
    from urllib import urlencode

_logger = logging.getLogger( name )


  class Response(object):
  """Holds the response from an API call."""
                      def init (self, response):
                      """
                      :param response: The return value from a
                      open call
                      on a urllib.build_opener()
                      :type response: urllib response object
                      """
                      self._status_code = response.getcode()
                      self._body = response.read()
                      self._headers = response.info()

 @property
```

```
    def status_code(self):
    """
    :return: integer, status code of API call
    """
    return self._status_code


    @property
    def body(self):
    """
    :return: response from the API
    """
    return self._body



@property
```

```python
    def headers(self):
    """
    :return: dict of response headers
    """
    return self._headers


        @property
        def to_dict(self):
        """
        :return: dict of response from the API
        """
        if self.body:
        return json.loads(self.body.decode('utf-8'))
        else:
        return None



class Client(object):
"""Quickly and easily access any REST or REST-like API."""
 # These are the supported HTTP verbs
```

```
methods = {'delete', 'get', 'patch', 'post', 'put'}

  def init (self,
  host,
  request_headers=None,
  version=None,
  url_path=None,
  append_slash=False,
        timeout=None):
        """
        :param host: Base URL for the api. (e.g.
https://api.sendgrid.com)
        :type host: string
        :param request_headers: A dictionary of the headers you want
                                           applied on all calls
                              :type request_headers: dictionary
                              :param version: The version number of the
                              API.
                              Subclass _build_versioned_url for custom
behavior.
                              Or just pass the version as part of the URL
                              (e.g. client._("/v3"))
                              :type version: integer
                              :param url_path: A list of the url path
                              segments
                              :type url_path: list of strings
                              """
                              self.host = host
        self.request_headers = request_headers or {}
        self._version = version
        # _url_path keeps track of the dynamically built url
        self._url_path = url_path or []
```

```
                              # APPEND SLASH set
                              self.append_slash = append_slash
                              self.timeout = timeout
def _build_versioned_url(self, url):
```

```
            """Subclass this function for your own needs.
            Or just pass the version as part of the URL
            (e.g. client._('/v3'))
            :param url: URI portion of the full URL being requested
            :type url: string
            :return: string
            """
            return '{}/v{}{}'.format(self.host, str(self._version),
            url)
```

```
    def _build_url(self, query_params):
    """Build the final URL to be passed to urllib


    :param query_params: A dictionary of all the query
```

parameters

```python
        :type query_params: dictionary
        :return: string
        """
        url = ''
        count = 0
        while count < len(self._url_path):
        url += '/{}'.format(self._url_path[count])
        count += 1

        # add slash
        if self.append_slash:
        url += '/'

        if query_params:
        url_values = urlencode(sorted(query_params.items()), True)
        url = '{}?{}'.format(url, url_values)

        if self._version:
        url = self._build_versioned_url(url)
        else:
        url = '{}{}'.format(self.host, url)
        return url

    def _update_headers(self, request_headers):
    """Update the headers for the request


    :param request_headers: headers to set for the API call
```

```
        :type request_headers: dictionary
        :return: dictionary
        """
        self.request_headers.update(request_headers)
```

```python
    def _build_client(self, name=None):

        """Make a new Client object

        :param name: Name of the url segment
        :type name: string
        :return: A Client object
        """
        url_path = self._url_path + [name] if name else self._url_path
        return Client(host=self.host,
                      version=self._version,
                      request_headers=self.request_headers,
                      url_path=url_path,
                      append_slash=self.append_slash,
                      timeout=self.timeout)

    def _make_request(self, opener, request, timeout=None):
        """Make the API call and return the response. This is separated into
        it's own function, so we can mock it easily for testing.

        :param opener:
```

```
:type opener:
:param request: url payload to request
:type request: urllib.Request object
:param timeout: timeout value or None
:type timeout: float
:return: urllib response
"""
timeout = timeout or self.timeout
try:
return opener.open(request, timeout=timeout)
except HTTPError as err:
exc = handle_error(err)
exc. cause = None
_logger.debug('{method} Response: {status}
```

```
    :return: string, version
    """
    self._version = args[0]

    return self._build_client()
    return get_version

# We have reached the end of the method chain, make the
API call
if name in self.methods:
method = name.upper()

        def http_request(
        request_body=None,
        query_params=None,
        request_headers=None,
        timeout=None,
        **_):
        """Make the API call
        :param timeout: HTTP request timeout. Will be
        propagated to
        urllib client
    :type timeout: float
```

```
        :param request_headers: HTTP headers. Will be merged into
        current client object state
        :type request_headers: dict
        :param query_params: HTTP query parameters
        :type query_params: dict
        :param request_body: HTTP request body
        :type request_body: string or json-serializable object
        :param kwargs:
        :return: Response object
        """
        if request_headers:
```

```
            self._update_headers(request_headers)

        if request_body is None:
            data = None
        else:
            # Don't serialize to a JSON formatted str
            # if we don't have a JSON Content-Type
            if 'Content-Type' in self.request_headers and \
                    self.request_headers['Content-Type'] != \
                    'application/json':
                data = request_body.encode('utf-8')
            else:
                self.request_headers.setdefault(
                    'Content-Type', 'application/json')
                data = json.dumps(request_body).encode('utf-8')

        opener = urllib.build_opener()
        request = urllib.Request(
            self._build_url(query_params),
            headers=self.request_headers,
            data=data,
        )
        request.get_method = lambda: method

        _logger.debug('{method} Request: {url}'.format(
            method=method,
            url=request.get_full_url()))
        if request.data:
            _logger.debug('PAYLOAD: {data}'.format(
                data=request.data))
        _logger.debug('HEADERS: {headers}'.format(
            headers=request.headers))

```

```
                response = Response(
                self._make_request(opener, request,
    timeout=timeout)
                    )

                      logger.debug('{method} Response: {status}
    {body}'.format(
                        method=method,
                        status=response.status_code,
                        body=response.body))

                    return response

        return http_request
        else:
            # Add a segment to the URL
            return self._(name)

        def getstate (self):
        return self. dict

    def setstate (self, state):
```