```python
#general imports
import os
from flask import session
import uuid
from dateutil import parser
# newsApi imports
from newsapi.newsapi_client import NewsApiClient
my_api_key = os.environ.get("API_KEY")
newsapi = NewsApiClient(api_key=my_api_key)
#async imports
from multiprocessing.dummy import Pool as ThreadPool



"""Individual functions for handling API response data"""
def collect_results(articles):
    results = []
    for article in articles:
        headline = article['title']
        source = article["source"]["name"]
        if not article["content"]:  # sometimes Newsapi is unable to provide content for each story
            content = "No content preview found. Click the link above to access the full story."
        else:
            content = article['content']
        author = article['author']
        description = article['description']
        url = article['url']
        image = article['urlToImage']
        api_date = article['publishedAt']
        published_at = parser.parse(api_date)
        id = uuid.uuid4().hex[:10]
        story = {'headline':headline, 'source':source, 'content':content,
        'author':author, 'description':description, 'url':url,
        'image':image, 'published_at':published_at, 'id': id}
        results.append(story)
    return results

def save_to_session(articles):
    """Saves results from api calls as a list of session objects"""
    if "results" in session: # clears previous session results if they exist
        session.pop('results')
    results  = collect_results(articles)
    session["results"] = results
    return results



"""Individual functions for separate types of API Calls"""
def cat_calls(query, slideshow = True):
    """API call to get generalized headlines for a specific catagory"""
    data = newsapi.get_top_headlines(language="en", category=f"{query}")
    articles = data['articles']
    if slideshow: #if api request is being made for homepage, transfer data directly rather than save to
session
        data = collect_results(articles)
        return data
    saved = save_to_session(articles)
    return saved

def top_headlines_call():
    """API call to get top headlines for all categories"""
    data = newsapi.get_top_headlines(language="en")
    articles = data['articles']
    saved = save_to_session(articles)
    return saved


def simple_search_call(query):
    """API call to get results from single search query"""
    data = newsapi.get_everything(q=f"{query}")
    articles = data['articles']
    spliced = articles[:10]
    saved = save_to_session(spliced)
    return saved


def advanced_search_call(query):
    from_ = str(query['date_from'])
    to = str(query['date_to'])


    if to == 'None' and from_ == 'None':
        data = newsapi.get_everything(q=f"{query['keyword']}", sources=f"{query['source']}",
language=f"{query['language']}", sort_by=f"{query['sort_by']}"
                                    )
    elif to == 'None' and from_ != 'None':
        data = newsapi.get_everything(q=f"{query['keyword']}", sources=f"{query['source']}",
language=f"{query['language']}", sort_by=f"{query['sort_by']}", from_param=f"{from_}"
                                    )
    elif to != 'None' and from_ == 'None':
        data = newsapi.get_everything(q=f"{query['keyword']}", sources=f"{query['source']}",
language=f"{query['language']}", sort_by=f"{query['sort_by']}", to=f"{to}"
                                    )
    else:
        data = newsapi.get_everything(q=f"{query['keyword']}", sources=f"{query['source']}",
language=f"{query['language']}", sort_by=f"{query['sort_by']}", from_param=f"{from_}", to=f"{to}"
                                    )
    # api seems to not want to allow dates to be optional if specified
    # I ran into trouble with the pagesize parameter from news-api, however a
    # temporary solution to this is to extract that number from the query dict,
    # and then splice the resulting list of articles.
    quantity = int(query['quantity'])
    articles = data['articles']
    spliced = articles[:quantity]
    saved = save_to_session(spliced)
    return saved


"""Executes Asyncronous API requests for cat_calls"""
def async_reqs(query):
    pool = ThreadPool(10)
    results = pool.map(cat_calls, query)
    pool.close()
    pool.join()
    return results
```