```
import asyncio
from bs4 import BeautifulSoup
from futures3 import ThreadPoolExecutor, as_completed
from io import StringIO, BytesIO
import json
import os
from pyppeteer import launch
import requests
from requests.adapters import HTTPAdapter
from requests.packages.urllib3.util.retry import Retry

from utils.fms import get_registry_inputdir_path, get_supplemental_inputdir_path, unzip


def extract_with_request(url, filename, http="get", headers=None, data=None, cookies=None, registry_code="", save=True,
                         is_test=False):
    """
    Extracts aircraft registry input data from remote source using a parameterized request

    :param (str) url: resource location of registry input data to extract
    :param (str) filename: name of output file to write extracted registry input data to
    :param (str) http: http method to use in request
    :param (dict) headers: header key-value pairs to use in request
    :param (dict) data: payload key-value pairs to use in request
    :param (dict) cookies: cookie key-value pairs to use in request
    :param (str) registry_code: registry-code specifying data or test sub-directory to write 'filename' argument in
    :param (bool) save: if 'True' writes extracted registry input data
    :param (bool) is_test: if 'True' the session is in a testing-context
    :return: registry input data extracted from remote source
    """

    if isinstance(data, dict):
        # stringify request payload
        data = json.dumps(data).encode()

    # initialize request
    session = requests.Session()
    adapter = HTTPAdapter(max_retries=Retry(total=25, backoff_factor=0.1))
    session.mount("https://", adapter)

    # extract registry input data from remote source
    response = getattr(session, http.lower())(url, headers=headers, data=data, cookies=cookies)
    response = response.content

    if save:
        # write extracted registry input data
        write_extracted_data(filename, registry_code, response, is_test)

    # format extracted registry input data to facilitate pandas integration
    response = format_extracted_data(filename, response)

    return response


def extract_with_headless_browser(url, from_id_crawler=False):
    """
    Extracts registry input data from remote source using a headless Chrome browser

    :param (str) url: resource location of registry input data to extract
    :param (bool) from_id_crawler: if 'True' then function was called from a thread spun by extract_with_id_crawler()
    :return: registry input data extracted from remote source
    """

    async def run_headless_browser(target):
        """
        Launches headless Chrome browser and fetches the html for the target website

        :return: html of target website
        """

        options = {"waitUntil": "networkidle0"}
        browser = await launch(handleSIGINT=False, handleSIGTERM=False, handleSIGHUP=False)

        page = await browser.newPage()
        await page.goto(target, options=options)

        html = await page.content()

        await page.close()
        await browser.close()

        return html

    if not from_id_crawler:
        # handle when running in the main thread
        response = BeautifulSoup(asyncio.get_event_loop().run_until_complete(run_headless_browser(url)), "html.parser")
    else:
        # handle when running in a sub-thread
        asyncio.set_event_loop(asyncio.new_event_loop())
        response = BeautifulSoup(asyncio.get_event_loop().run_until_complete(run_headless_browser(url)), "html.parser")

    return response


def extract_with_id_crawler(urls, filename, http="get", headers=None, data=None, cookies=None, registry_code="",
                            save=True, is_test=False):
    """
    Extracts and aggregates aircraft registry input data from remote sources using parameterized multi-threaded requests

    :param (list) urls: resource locations of registry input data to extract
    :param (str) filename: name of output file to write extracted registry input data to
    :param (str) http: http method to use in request
    :param (dict) headers: header key-value pairs to use in request
    :param (dict) data: payload key-value pairs to use in request
    :param (dict) cookies: cookie key-value pairs to use in request
    :param (str) registry_code: registry-code specifying data sub-directory to write 'filename' argument in
    :param (bool) save: if 'True' writes extracted registry input data
    :param (bool) is_test: if 'True' the session is in a testing-context
    :return: registry input data extracted from remote sources
    """

    # initialize storage for aggregating registry input data
    response = []

    # initialize and start multi-threaded processing
    with ThreadPoolExecutor() as executor:
        # extract registry input data from remote sources
        if not filename.endswith(".html"):
            futures = (executor.submit(
                extract_with_request, url, filename, http, headers, data, cookies, registry_code, False
            ) for url in urls)
        else:
            futures = (executor.submit(
                extract_with_headless_browser, url, True
            ) for url in urls)

        for future in as_completed(futures):
            result = future.result()

            if result is not None:
                # aggregate extracted registry input data
                response.append(result)

    if save:
        # write extracted registry input data
        write_extracted_data(filename, registry_code, response, is_test)

    # format extracted registry input data to facilitate pandas integration
    response = format_extracted_data(filename, response)

    return response


def format_extracted_data(filename, response):
    """
    Unpacks and formats extracted registry input data based on content type to facilitate pandas integration

    :param (str) filename: name of output file to write extracted registry input data to
    :param response: extracted registry input data returned from an http request
    :return: unpacked and formatted registry input data extracted from remote source
    """

    if filename.endswith(".zip"):
        response = None
    elif filename.endswith(".pdf") and isinstance(response, bytes):
        response = BytesIO(response)
    elif filename.endswith(".json"):
        while isinstance(response, (str, bytes)):
            try:
                response = json.loads(response)
            except json.decoder.JSONDecodeError:
                response = None
    elif filename.endswith(".html"):
        if isinstance(response, list):
            response = "\n".join([str(x) for x in response])
        elif isinstance(response, bytes):
            response = response.decode()
    elif not filename.endswith(".xlsx") and not filename.endswith(".xls") and isinstance(response, bytes):
        try:
            response = StringIO(response.decode())
        except UnicodeDecodeError:
            response = StringIO(response.decode(encoding="ansi"))

    return response


def write_extracted_data(filename, registry_code, response, is_test=False, is_supplemental=False):
    """
    Writes extracted registry input data to appropriate file location

    :param (str) filename: name of output file to write extracted registry input data to
    :param (str) registry_code: registry-code specifying data sub-directory to write 'filename' argument in
    :param response: extracted registry input data to write
    :param (bool) is_test: if 'True' the session is in a testing-context
    :param (bool) is_supplemental:
    :return: effect - creates [DATA_DIR|TEST_DIR]/[REGISTRY_INPUT_DIR|SUPPLEMENTAL_INPUT_DIR]/[registry_code]/[filename] asset
    """

    if not is_supplemental:
        root = get_registry_inputdir_path(is_test)
    else:
        root = get_supplemental_inputdir_path(is_test)

    path = os.path.join(root, registry_code.lower(), filename)

    if filename.endswith(".json"):
        while isinstance(response, (str, bytes)):
            response = json.loads(response)

        with open(path, "w") as file:
            json.dump(response, file)
    elif filename.endswith(".html"):
        if isinstance(response, list):
            response = "\n".join([str(x) for x in response])

        with open(path, "w") as file:
            file.write(response)
    else:
        if isinstance(response, bytes):
            with open(path, "wb") as file:
                file.write(response)
        else:
            with open(path, "w") as file:
                file.write(response)

    # extract zipped file into a new adjacent directory with the same name
    if filename.endswith(".zip"):
        unzip(path)
```