goal for this document

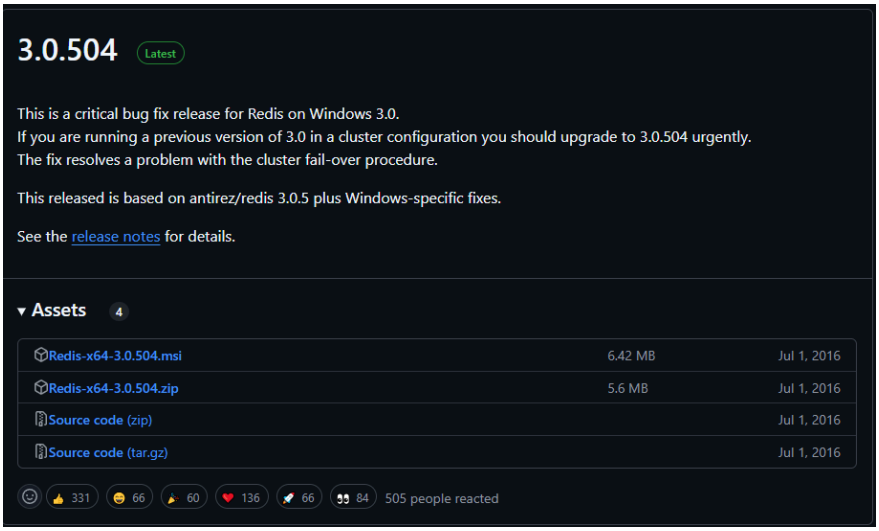
While testing our app, I noticed that many api calls were being made unnecessarily causing me to get rate limited. The spotify api has this thing where if a certain amount of api calls are made within 30 seconds, it slows you down a lot, which is no good. My initial thought was to store some of the user information, like their playlists and all the tracks in the playlist they select, in the session cookie, then query the session cookie when needed. I soon realized that this would cause the session cookie to become huge and when session cookies are huge, some of the information in them can be ignored by the browser, which is also no good! I did some research and found out about server side caching. Right now as the code stands, a lot of information is being stored in the session cookie, like the token information, user credential info, playlist information, and so on. This is not practical since session cookies are not built to store that much information (most browsers have a cookie size limit of 4kb, very tiny). Server side caching does exactly what the name entails and takes some of the weight off of the session cookie. With all that out of the way, my goal for this document is to implement server side caching and to reduce the amount of unnecessary api calls made.

server side caching

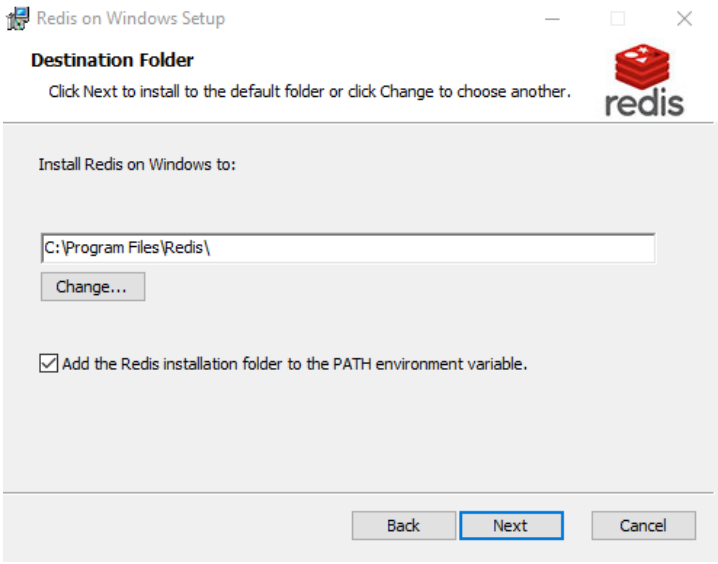
The tool I will be using for server side caching is redis. Redis is like a super fast notebook for an app, it stores data in memory rather than on a disk so information can be retrieved super fast. It works like a dictionary in the sense that data is stored as key-value pairs.

installing redis

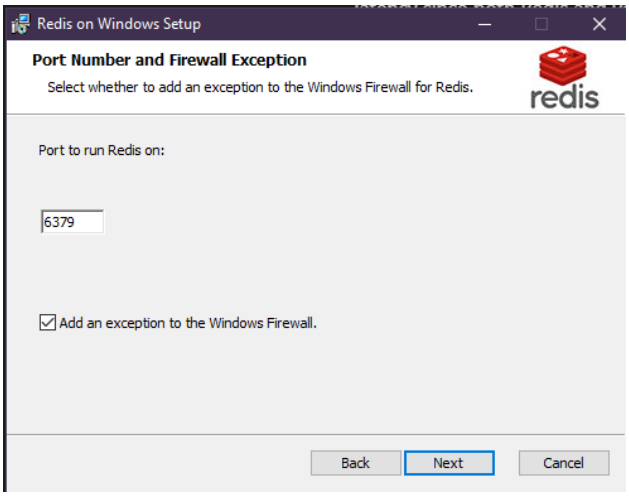
https://github.com/microsoftarchive/redis/releases



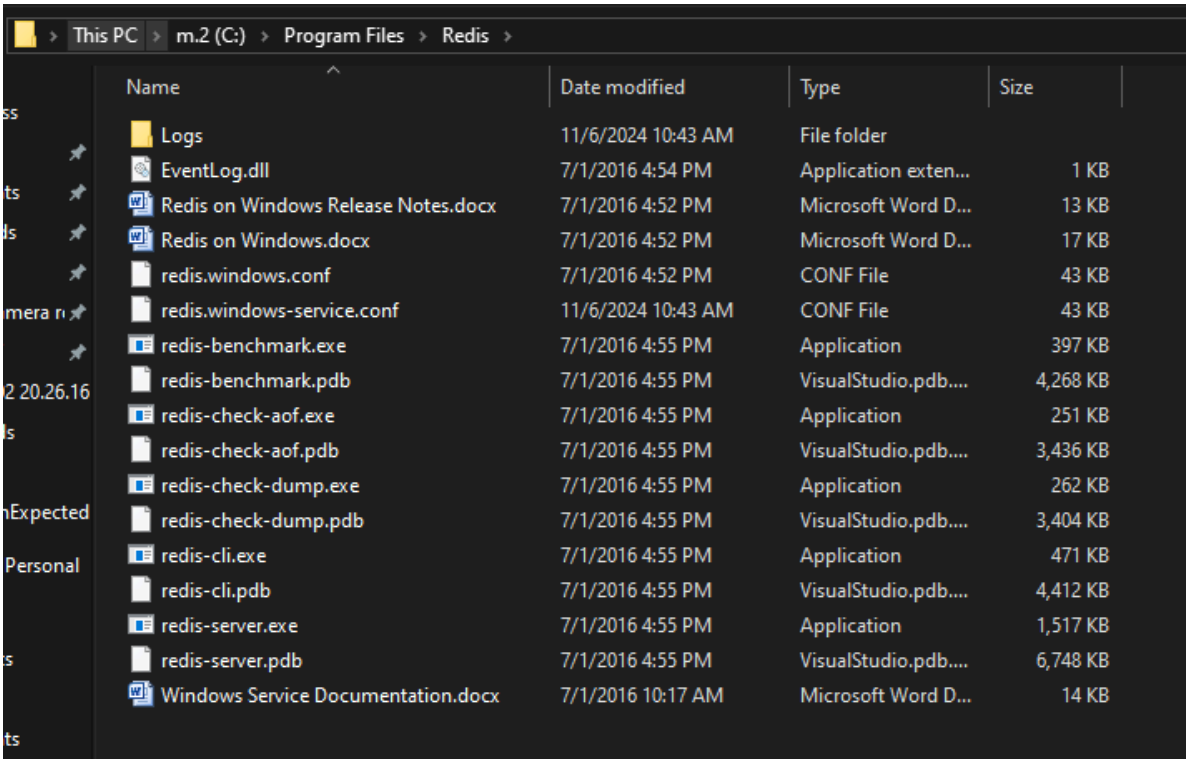
Download the .msi file and run.



Make sure to add redis installation folder to the path environment variable.

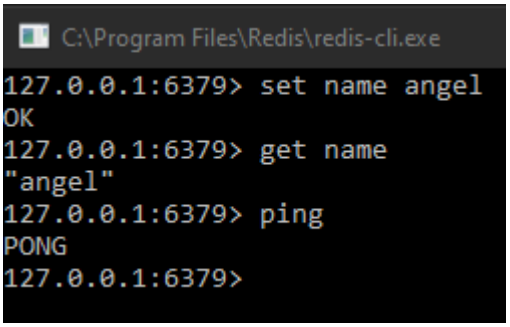


DO NOT CHANGE THE PORT NUMBER, keep it at default value.



After installing, run redis-cli.exe to initialize redis server.

Running redis-clie.exe brings up the following window



I did some test commands like set, get, and ping just to verify that the server is functioning properly.

implementing redis in app.py

step #1, import necessary modules (redis, json)

```
import redis
import json
```

step #2, connect to redis server

```python
#Connect to redis, make sure redis is running locally
redis_client = redis.StrictRedis(host='localhost', port=6379, db=0, decode_responses=True)
```

make sure to run redis-cli.exe before trying to run app.py

step #3, implement redis caching for get_all_tracks() function

```python
#function to fetch all tracks in a playlist
def get_all_tracks(sp, playlist_id):
    #check if tracks are cached in redis
    cached_tracks = redis_client.get(playlist_id)
    if cached_tracks:
        return json.loads(cached_tracks)

    offset = 0
    all_tracks = []

    #fetch tracks in batches, add to all_tracks until no more tracks are left
    while True:
        tracks = sp.playlist_items(playlist_id, offset=offset, limit=100)
        all_tracks.extend(tracks['items'])

        #stop if there are no more tracks to fetch
        if len(tracks['items']) == 0:
            break

        offset += len(tracks['items'])

    #cache tracks in redis with an expiration of 4 hours
    redis_client.setex(playlist_id, 14400, json.dumps(all_tracks))
    return all_tracks
```

- line 53: checks redis to see if the playlist's track data is already cached
- if cached data exists, redis returns a json encoded string containing all the track data for that playlist with json.loads(cached_tracks), this string is then decoded into a usable format. This makes it to where the app now checks redis for information, rather than making the api call every time information is requested.
- if cached data does NOT exist, None is returned indicating that data has not been cached or has expired. If this is the case, the function retrieves the playlist track data with spotify api calls, stores this data in redis, and sets it to expire in four hours (line 72, 14400 seconds = four hours) with redis_client.setex()

step #4, implement redis caching for get_all_playlists() function

```python
#by default, spotify api can only fetch 50 playlists per request
#function to get all user playlists(>50 playlists)
def get_all_playlists(sp, username):
    #dynamic key for redis dictionary, key will include username
    dictionary_key = f"user_playlists_{username.replace(' ', '_')}"

    #check if playlists are cached in redis
    cached_playlists = redis_client.get(dictionary_key)
    if cached_playlists:
        return json.loads(cached_playlists)

    playlists = []
    offset = 0
    limit = 50

    while True:
        response = sp.current_user_playlists(offset=offset, limit=limit)
        playlists.extend(response['items'])

        #stop if all playlists have been fetched
        if len(response['items']) == 0:
            break

        offset += len(response['items'])

    #cache playlists in redis with an expiration of 4 hours
    redis_client.setex(dictionary_key, 14400, json.dumps(playlists))
    return playlists
```

- line 79 creates a dictionary key for the user's playlist, replacing any spaces in their usernames with underscores

- line 82 checks redis to see if the playlists are already cached
- if cached data exists, redis returns a json encoded string containing the playlists with json.loads(cached_tracks), this string is then decoded into a usable format.
- if cached data does NOT exist, None is returned indicating that data has not been cached or has expired. If this is the case, the function retrieves the playlists with spotify api calls, stores this data in redis, and sets it to expire in four hours (line 101, 14400 seconds = four hours) with redis_client.setex()