First Thoughts: Nov 28, 2024

# Design Document for Social Dance Calendar App

## The Problem

Whenever you start looking for who is doing what in social dancing, you have to go to a large number of sites. The information is often outdated, incorrect, etc. People that are in that local area know the drill and where to go on the internet, but somebody coming in cold from the outside, does not.

## The Solution

I want to take a list of key words and urls, provide those to a LLM to judge if there is content that is helpful for building this calendar. The results of this judging process returns a JSON file that is then stored in a series of JSON tables. Then the app provides a report / calendar with the names, locations, times, description, and cost for the events.

This actually 2 different applications that work together. The first is the one that goes out and gets the information and populates the sql tables. Lets call that Get\_Info. The second one is a web app that queries the database based on a chat interface on the web. Lets call that Display\_Info.

## The Environment

It will be a web app and will use SQL to store the data.

## The Design

### Get\_Info

#### Input

1. List of urls in a db table. This is human generated. This needs a key. The key is going to be dance style and city including state / province and country. An example table is urls.csv.
2. List of keywords in a db table.
   1. First column should be location (city state country). Second column should be the dance style, the third column should be the search term. For example salsa would be ‘salsa social dance events’ or what ever makes sense based on what we are searching for. For example, running could be ‘running clubs’.

At least one of the 2 inputs are required and preferably both.

#### Link Depth

It is likely that there will be links to other pages that we will want to include in our urls that we give to Clean Up. Only go 3 levels deep.

#### Thoughts

OpenAI was not great in terms of generating the urls. It was quite stochastic. Sometimes good, sometimes terrible. With the same prompt, Google was not good either. I am pretty suspicious that what I need to do is create a series of keywords and then use Google NOT OpenAI to generate those urls.

We should also be picking up special events like SwingCouver or local workshops. This may require us to go beyond the first page. I think we should take the top 20 hits for each line of keywords.

#### Facebook and Instagram

You will need to sign up for the groups to get access to the pages.

### Clean Up

#### The Judge

LLMs will decide if the content is helpful. From Input comes a list urls.

#### Dates

We only want pertinent dates. We do not want stuff from the past. We should default to today and one week from today. The user could put their own date range in if they want. We would use a LLM to parse that.

### Extract

Then the application will extract the information in the necessary format (JSON). Then write it to the SQL tables.

### Display\_Info

#### Output

A function will be written that creates the output report. This queries the above tables. It has the chat interface from which it generates the sql query.

# Requirements

## Project Context

### Target Audience

My audience is anybody in the world that is frustrated by the amount of time that they need to spend finding activities on the internet that they want to do, especially when they visit a new location. This person is also frustrated that they are not informed of events that would be of interest to them. I want to start with social dancing. That is West Coast Swing, Salsa, Bachata, Kizomba, and Zouk in the city of Victoria, British Columbia, Canada

### Level of Detail

I need implementation details. I am capable of programming in Python.

### Key Stakeholders

I, Lindsay Moir, a 69 year old male is the primary stakeholder. If this application is sufficiently accurate, comprehensive, and easy to use I will provide it to people initially in the Victoria, BC, Canada dance community to use. I will require it to make money at some point. There will be costs for hosting the application. I also need to generate some income and am getting fairly pessimistic that I will be able to find work in the data science area due to my age and preferences. I do not want to work a full time job.

## Functional Requirements:

Get\_Info

* As a system administrator, I want to start a job that looks at the urls that I give it plus the other urls that it finds during a google search and populate the SQL database with the latest information. This can take as long as 5 minutes to run.

Display\_Info

* As a user, I want to be able to chat with a web interface.
* Initially the application (Display\_info) will assume that their location that they are interested in is, the IP address that they are at. They will be asked to confirm that.
* Eventually during this chat process of the user inputting text and the LLM prompting the user, I will have the name of the city, the province, and the country. I will have the activity that he /she is interested in (currently dance for the initial implementation), and the dates and times that they want to see.
* Once I have this information, then the appropriate information will be given to the program to generate a sql query that answers the question and displays the results in calendar format.

## Non-Functional Requirements:

* The backend must maintain 99.9% uptime and encrypt all API communications.

## Technical Specifications:

* Must connect to a PostgreSQL database for writing, updating, and reading.

https://chatgpt.com/share/674e1151-ee00-8011-8faa-357171e897e0

**Finalized Requirements Update for Get\_Info**

**Expanded Functional Requirements**

1. **Initial URL Table Processing**:
   * For each URL in the **Initial URL Table**:
     + Evaluate its content for relevance using the LLM.
     + If relevant, extract event information and store it in the SQL database.
     + Log the results of evaluation, including metadata like timestamps, referring pages, and ranking of relevance (True, False, NaN).
2. **Keyword Search Integration**:
   * Perform a **Google search** using keywords from the **Keywords Table** in the SQL database.
   * Retrieve results from the **first two pages** of Google search results.
   * Check whether each resulting URL exists in the **URLs Table**:
     + **If it exists and is relevant**: Process as usual, extracting data and updating logs.
     + **If it does not exist**: Add the new URL to the URLs Table for further evaluation and crawling.
3. **Dynamic Crawling from Search Results**:
   * For newly discovered URLs from the Google search:
     + Evaluate the page for relevance using the LLM.
     + If relevant, extract event information, store it in the database, and crawl hyperlinks on the page to evaluate connected pages.
     + Log all evaluations, including relevance ranking and errors (if any).
4. **Relevance Criteria**:
   * The LLM evaluates each page using a prompt designed to detect event details specific to dance events (e.g., event name, location, time, cost).
   * URLs will be ranked as:
     + **True (Useful)**: Contains relevant event details.
     + **False (Not Useful)**: Contains no relevant data.
     + **NaN (Nonexistent)**: Invalid or broken links.
5. **Database Updates**:
   * Newly relevant URLs from Google search and crawled pages are appended to the **URLs Table** for inclusion in future runs.

**Updated Scheduling and Automation**

* The system will continue to run daily at **12:01 AM**, performing the following tasks:
  1. Process existing URLs in the **Initial URL Table**.
  2. Perform Google searches using keywords from the **Keywords Table**.
  3. Evaluate and update all results (existing and new) in the SQL database.

**Technical Specifications**

1. **Database Design Updates**:
   * **Tables**:
     + **Event Data Table**: Stores extracted event details (e.g., name, location, date, time, description, cost, source URL).
     + **URL Metadata Table**: Logs details about each URL (e.g., address, status, relevance score, last visited date, referring URL, visit count, errors).
     + **Keywords Table**: Stores keywords for Google searches (e.g., "West Coast Swing Victoria," "Bachata events BC").
2. **Frameworks and Tools**:
   * **Google Search Integration**: Use tools like SerpAPI or Google Programmable Search Engine to automate keyword searches and retrieve result URLs.
   * **Web Scraping and Crawling**: Continue to use **Scrapy** for crawling and evaluating linked pages.
   * **LLM Evaluation**: Use OpenAI or similar LLMs to determine the relevance of page content dynamically.
3. **Error Handling and Resilience**:
   * Record all failed attempts (e.g., broken links, invalid pages) and retry if transient errors occur.
   * Log search and crawling performance metrics to evaluate system coverage and accuracy over time.

**New Logging Requirements**

1. **Search Logging**:
   * Log each Google search with details such as:
     + Keywords used.
     + URLs returned by the search.
     + URLs processed (new or existing).
     + Timestamp of the search.
2. **Resilience Logging**:
   * Track and log how the system adapts to changes in website structures or invalid pages.

**Non-Functional Requirements**

* Maintain modularity to support switching search engines or LLM providers as needed.
* Use redundancy in the crawling process to account for the stochastic nature of LLM evaluations.
* Prioritize user privacy and encrypt sensitive communications.

**Future Considerations**

* **Scalability**: Adapt to handle thousands of new URLs and keywords as the application expands to new regions or activity types.
* **Enhanced Ranking Metrics**: Introduce machine learning models to rank relevance scores more effectively based on historical performance.
* **Customization**: Allow users to input additional keywords or URLs directly to enhance search coverage.

This update ensures the **Get\_Info** system dynamically discovers and adapts to new content while maintaining a robust logging and evaluation mechanism. Let me know if additional refinements are required!

# Dec 2, 2024

I have started coding the scrapy.py script. It works but it needs the following improvements.

Retrieving only URLS and Has a Poor Relevance Algorithm

It is just looking for keywords in the urls. I want the following process.

1. I do want all of the urls on the page. Scrapy is doing that now. So, this is good.
2. For each url, Scrapy needs to get ALL of the source.
3. Have the LLM figure out whether or not this is a page that is applicable to people that want to “dance”. Eventually we will want this to loop thru all of the keyword search terms but for now, lets just hard code what we are looking for. We should be using salsa.
4. If it is a page that is applicable, see if there are any events on the page. If there are return those events in JSON format (data, time, location, description, cost, url)

## Need to Fix

1. I mark an url as relevant based on keywords. However, 99% of the time, if there is no event there the first time you look at a relevant url, on subsequent visits it is likely that there will not be any events. So, it does not make a lot of sense to burden the application with constantly going back to these urls.
   1. Perhaps once every 3 months depending on how much processing we have to do. This update cycle should probably be adjustable in the config file.
   2. Anyways, I need to figure out how to do this. It is not difficult. Probably just need to put the code in for when it successfully finds an event to put the timestamp in a column for that. Probably best to do that on urls.
      1. The issue is that the urls table may be growing quite quickly now. I guess I could simply alter the table using pgadmin to add the column and adjust the code from there?
2. We have a timestamp when we create the url. We do not have one when we update it. We should change the table to include a time\_stamp\_updated. Pretty easy just add the column and when you update the url or write to the url table, just add the column / update it. So, you will have a created timestamp and an updated timestamp columns.
3. I have inconsistent column names between the urls and events tables. One if plural the other is not. These should be consistent in the code and in the tables. I think it would be better to make them all singular. Think about this. I suggest you ask ChatGPT.
4. # Check if the URL contains 'login' or groups not in url
   1. # I am pretty sure this is redundant but I will leave it here for now
   2. if 'login' in url or '/groups/' not in url:
   3. logging.info(f"def parse(): URL {url} marked as irrelevant due to Facebook login link.")
   4. update\_url(url, update\_other\_links='No', relevant=False, increment\_crawl\_trys=0)
   5. return False

# fb.py

I have decided to put the facebook code in a separate .py file. The other one is just getting too big. I have some code now that does a good job of getting the event links off of face book using the search function. Right now that search function is just taking an url that I have given it as a string.

1. However, what I should be doing is having some automated way of creating an url that is sent to this function called fb\_search\_url(). I should think about how I want to do this.
2. I also need some recursion on this. The smart thing to do would be to let scrapy do this. So, I need to ask chatgpt to set that up. I only want 3 levels on it.

# db.py

I also should get the database code out of scraper.py and put it into db.py. This whole thing is just a little unwieldly right now. I need to think about the issue of importing this in. Come to think of it that is not problem since they are all functions that are not in classes.

# Updated Scrape Pattern Jan 2, 2025

## Config table

1. We will need to keep run information. I suggest that we snap shot the config file. Pretty sure that yaml creates a dictionary that is dead easy to create a dataframe and then put that into a config table with a time\_stamp. We will use that for post processing statistics and machine learning.

## Multiple Scrapers

We will have multiple scraper passes based on a variety of criteria.

### High Value URLS

1. First pass is to use urls.csv. This is a list of urls that we think are high value urls where most of the events are. This will use scrapy and go 3 levels deep. It will get all of the easy to find events on those pages (including calendar events) and will go 3 levels deep on those websites. This was really my first implementation.
2. Post this run, we will have those urls in the urls table. What makes them unique is that they will have a .com suffix. So, this means that when we read in urls.csv we have to make sure that we capture the original url and store it. Then later on when we want to rerun and look at those website we do no need to be dependent on the urls.csv file. So, we could call this file the seed urls for a particular location. Later on, we need to automate this and allow people to suggest locations and seed urls which will go into a table called seed\_urls.

### Google Search

Once we have run the High Value URLS pass, then we need to run the google search pass.

1. This will return the titles and urls of all of the keywords. This needs to be done in a loop.
2. Once we run this loop then we need to see what titles and urls we already have.
3. If we already have the url, then that is easy. We need to check the time\_stamp of that event.
   1. Then we will check the time\_stamp of all matching urls and see when they were last updated. There is a time\_stamp of the urls table (creation date). There is a time\_stamp on the events table. There is obviously a current time. The algorithm will be:
      1. If the time\_stamp of the event\_table (last time it was updated) is older than14 days, then check and make sure all of the information is current. This 14 day period will be in the config file and is adjustable. Unfortunately, no way to do that other than just scrape it again. We should compare the 2 versions though. We need to come up with stats on how often this event information is changing. If it does change, then delete the event from the events table and put it into the history\_events table. It will just be a simple delete on one and an insert in the other.
         1. I thought of leaving this in the events table and simply using the timestamp after writing it again, however the events table will soon become polluted with an enormous amount of outdated information. We could, use the idea of a base event and then apply incremental changes to it. That would cut down on the amount of redundant information but … storage is cheap, processing time is relatively more expensive.
      2. As the date gets closer it becomes more important to have the latest information. So, if the data is less than 7 days away then you always check and see if the information has changed as above. We are running this scraper daily so, … this might get big very fast.
4. There is a maximum of 100 requests per day. If we restrict it to 5 results, then that is 500 potential matches.
5. We need to reduce the number of keywords. If we say kizomba (for example) it is going to bring up urban kiz (for example) anyways. Come to think of it, we can have 2 sets of keywords. One is for google. That would be a little grosser and then we could ask the keyword matching to be more flexible and the LLM to be exact. So, 3 versions of keywords. We need a keywords table in the database. The columns would be:
   1. topic – in this case we are going to call it social dance
   2. search\_engine
      1. Facebook for example is going to run down my social graph. I ran a lot of tests. It does not work that well at all. We can just use “dance” as the keyword when coming at it thru facebook search. My take is that this is a very limiting way to get into facebook and ultimately may not be worth using. “dance” does not do much either. I guess if we want we can run it but … I do not expect much to come from facebook search.
      2. Google it will be these keywords
         1. bachata, cha cha cha, kizomba, merengue, salsa, swing, tango, west coast swing
      3. Probably All In Events also
   3. text\_keywords
      1. latin - bachata, kizomba, merengue, salsa,
      2. kizomba -kiz, kizomba, semba, tarraxo, tarraxa, tarraxinha, urban kiz,
      3. swing - balboa, east coast swing, lindy hop, swing
      4. west coast swing – wcs, west coast swing,

### Facebook Search

We want to run an experiment on the facebook search. We want to see if the event urls correspond exactly to the scrape order on the page with the exact same number of links. If they are the same, then … we can stop screwing around with the current system.

Prompt Choosing

There are 3 possible prompts

Facebook prompt – This will be used when you have facebook groups or any other facebook page whereby you want to get ALL of the events mentioned regardless of whether or not they have an url associated with it. Llm.generate\_prompt() is going to get another variable called prompt\_type. For this one it is ‘fb’

Facebook single prompt – This will be used when you have googled a row in the database and the best solution is to use Facebook. In this case we do NOT want multiple events. We ONLY want the first event. I AM PRETTY SURE WE NEED TO BUILD THIS PROMPT AND PROPERLY DO SOME REGEX WORK ON THIS IN A FUNCTION. Llm.generate\_prompt() expects ‘single \_event’

Default – This is used for everything else until such time as we believe we need a separate prompt for example All Events In or Event Brite. Llm.generate\_prompt() expects ‘default’.

## Address Requirements

I have a working version of something that parses addresses (location column in events table). However, it needs to be integrated in.

## January 1, 2025

Need to run the entire workflow. I may be close to having a working back end!!

## Requirements For Update Events

I have thought about this. The right thing to do is to make this a single pass. By that I mean, we should NOT update. We should drop the events table and build it out each time. It is too difficult to figure out what is current or what is not. Lets just run with what we know is current. We should however, post process and create some helpful tables that can fill out the events. Not even sure if we need an event\_id. Lets leave it, you almost always need a primary key.

We need an organization table. This would have org\_id, org\_name, web\_url, fb\_url, ig\_url, phone, email, and address\_id. We should build a program that scrapes for this information, based on a group by of the org\_name in events. It should run weekly and just try to keep that information current. I need to write out the specs for this better.

## Duplicate Checking

Pretty hard to check all of the events against every other event. That is a little ridiculous. However, when the users hit the database, it will be very annoying if they get like 10 obvious duplicates of the same event up. Yet, if they are only asking for 10 or 20 rows at a time (likely), it is computable to check all against each other and choose one above all and remove the duplicates. We can probably do that with above 99% accuracy.

To do this, I would suggest using fuzzy wuzzy. We could get a fuzzy wuzzy score by column and then average over the columns. We would not have to choose the columns then. When we got 2 that scored over 90 (on config and and could be adjusted) on the fuzzy wuzzy score, we would choose the one that had the most recent time stamp. HOWEVER, for the one that was discarded and there were columns not filled in on the one that we were keeping, we would use the columns from the discarded event to update the kept event with those values.

Bottom line is we allow duplicates because we run the update every night at 12 midnight

## Truncate Description Column in Events

I decided to not truncate. There is lots of good information in the description. All that we will do in the front end is only show 100 characters (for example) and if they want they can get more by asking for it. We can probably do one of those little mouse over things.

## Post Processing

Fix the lack of an address\_id in the events table. Some are null and they should not be. Pretty easy fix, just go thru find them and fix them. I can probably get chatgpt to write all of the code for me.

## Google Search

We should still do a general google search where we look for all of the events that might have escaped. This will also fill in missing columns in rows (hopefully). We would run all of the keywords thru. I am a little concerned that we might run out of daily freebies. I think we should do it according to dance style keywords and see what it comes up with. I think we can just go with the hits on the first page.

## All Events In

We could also do one for this search engine. I think it is a good idea. Probably some functions that we could use across All Events In and Event Brite

## Event Brite

Same here. Probably a good idea. Keep it simple. Do one for every search engine.

# January 12, 2025

## Google Search – gs.py

When we ran it with all of the keywords we ended up with 33 urls. That is not bad. I think that is workable. It looks like pretty good coverage and what I would expect.

The process for determining if the url is worth scraping is as follows.

1. If it is social\_media (Facebook, Instagram) write it to the fb table. You need to check and see if it is there already though.
2. See if the dance\_style column from the keywords .csv file is in the title, url, or snippet

# January 15, 2025

gs.py needs to be ran BEFORE scraper.py. gs.py outputs a .csv file that needs to be included when scraper.py runs from .csv’s.

Design for Eventbrite search engine and url handing

## January 17, 2024

I am working on the design to deal with no org\_name. Should we delete the events and if so based on what criteria and then what should we do with the remaining events.

So far deletion looks like this.

1. No org\_name, dance\_style, or url BUT if it has an address\_id keep it

Done

Now duplicates looks like this.

1. We find duplicates by:
   1. Sorting all events by start\_date, start\_time
   2. If 2 or more events have the same start\_date, start\_time then:
      1. Fuzzy the name of the event
2. If there is a duplicate that gets a fuzzy score of more than 80 for event\_name then you want to keep one of the rows. In order of decision:
   1. The one that has an url, you keep it.
   2. If all do not have urls, then count up the filled columns. For example row 1 has 5 columns filled in and row 2 has 7 columns filled in, then you keep column 7.
   3. For any columns that are not filled in on the kept row and there are columns in the to be dropped row that are not present in the kept row, have the kept row get updated by those columns from the to be dropped row.

## Location Based Dropping

Drop events that are outside of BC. That includes all events that are in the USA or international. Need to be careful here, if the country field is blank leave it. If it say anything other than null or blank or CA then delete that event

## Likely Drops

1. dance\_style and url are == ‘’ AND event\_type == ‘other’ AND location IS NULL and description IS NULL

## No URL Treatment

1. Get all of the events that do not have an url.
2. If url: continue
3. else: logging.debug(‘all events should have had an event\_name’)
   1. Get the next url in the for loop.
4. Run a google search using the event\_name and see if the title fuzzy matches (80%) with the event\_name. Some of the google search titles will match. You want all of them that match. You only want to select one. In order of preference it is:
   1. url that contains ‘alleventsin’, then ‘eventbrite’, any other url that matches and if the only on that matches contains ‘facebook’ or ‘instagram’ us that one.
5. Then use that url to fill in the event for that row.
   1. That means using extract\_text\_with\_playwright(url) which is a method in class EventSpider() which should be instantiated as es\_instance(). This method returns extracted\_text.
6. Now that you have the extracted\_text you can run llm\_handler.check\_keywords\_in\_text((url, extracted\_text, org\_name, keywords\_list). This returns a True if there are keywords in the extracted\_text or a False if there are no keywords in the extracted\_text.
7. If True:
   1. # Generate prompt, query LLM, and process the response.
   2. prompt = self.generate\_prompt(url, extracted\_text, 'default')
   3. llm\_response = self.query\_llm(prompt, url)
   4. if llm\_response:
   5. parsed\_result = self.extract\_and\_parse\_json(llm\_response, url)
   6. if parsed\_result:
   7. events\_df = pd.DataFrame(parsed\_result)
   8. Make sure that events\_df is just one row. If it is more than one row, compare the event\_name in the original row to the event\_name in events\_df using fuzzy. One of these rows should match at least at 80 from fuzzy. That is the one that you want and will be referred to as the new row from here on in in this explanation.
8. Now that you have events\_df, you can compare the row from events that started this and replace the original information in the row (original row) that you are working on as follows.
   1. Assemble the following information into a new row that replaces the original row.
      1. For each column in the original\_row compare that to the new row returned from event\_df. If the original\_row is == ‘’ or NULL then take the information from the new row and put that information in the column for the original row.
      2. If they both have information count the length of characters in the original row and the new row. Whichever has more characters that information is now used in the new row.
      3. Write the new row to the events table replacing the original row.
9. Update / write the url to the urls table. Use write\_url\_to\_db(self, org\_names, keywords, url, other\_links, relevant, increment\_crawl\_trys) to do this. You can get org\_names, keywords (called dance\_style in events\_df) from events\_df. other\_links = ‘’, and relevant=True, increment\_crawl\_trys = 1. This method is in the DatabaseHandler class and should be instantiated in this program in the imports using:

from db.py import DatabaseHandler. The instance is db\_handler().

1. It should be instantiated in the \_\_main\_\_ as db\_handler = DatabaseHandler(config)

Please provide me code for that above. The code should provide a docstring at the top of the file that describes what this code does.

## January 19, 2025

So, we are working on the refactoring and we have the problem that our friends at Facebook are screwing around with how they put text on the page. When we go looking to fix the missing url problem, we are going to get all of the text. There will be a number of events. We only want the event with the largest number of columns filled out. Then that one should get the url that we have been carrying around. The reason is that I have playwright click the see more button and we get more information. The other ones, there is not that option.

## Scraper.py

When we try to log onto google it refuses us because it does not trust Chromium. We may have to go with firefox.

## Processing Order To Load Database

Doing this on Feb 16, 2025

1. Config settings
   1. config[‘llm’][‘provider’]: Mistral is free as of Feb 16, 2025
   2. config[‘testing’][‘drop\_tables’]: True
2. db.py
   1. AFTER running config[‘testing’][‘drop\_tables’]: False
3. gs.py – Only creates a .csv
   1. Change to config[‘llm][‘provider]: openai for this one. I think google is too fast for mistral. It is complaining about rate limits. This is probably not true. I have been using them a lot and I think I just hit the limit
   2. Check and make sure there is something in that .csv after you run this.
   3. config[‘llm’][‘provider’]: openai AND the model should be:
   4. config[‘llm’][‘model’]: ‘GPT-4o mini’ This is a VERY simple prompt.
4. ebs.py
   1. Only writes to the database
5. emails.py
   1. Likely that the oauth screen will come up. Can’t run this headless.
6. try config[‘crawling’][‘headless]: True
7. rd\_ext.py
   1. reads config[‘input’][‘’edge\_cases’] a .csv with odd urls. Check if first before running.
   2. Writes to the database
8. scraper.py
   1. Likely that try config[‘crawling’][‘headless]: True will need to be False
9. fb.py
   1. Need to change config[‘crawling’][‘headless’]: False. Facebook is just a pain in the ass.
10. db.py
    1. to dedup the database
11. clean\_up.py
12. dedup\_llm.py
13. app.py
    1. Check and make sure it is running fine on Render
14. Main.py
    1. Check and make sure it is running fine on Render.