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 The information is often outdated, incorrect, etc.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.
    a. 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!

Processing Order To Load Database

Pipeline

1. events_table_backup_and_drop
2. db.py
3. emails.py
    a. Likely that the oauth screen will come up. Run it first.
    b. This way we can deal with the oath screen if we do this first.
    c. try config[‘crawling’][‘headless]: True
4. gs.py – Only creates a .csv
    a. Delete the .csv
    b. Check and make sure there is something in that .csv after you run this.
5. ebs.py
    a. preprocessing need to set headless=True
    b. Only writes to the database
6. rd_ext.py
    a. reads config[‘input’][‘’edge_cases’] a .csv with odd urls. Check if first before running.
    b. Writes to the database
7. scraper.py
    a. Likely that try config[‘crawling’][‘headless]: True
8. fb.py
    a. Need to change config[‘crawling’][‘headless’]: False. Facebook is just a pain in the ass.
9. backup_db
    a. Takes a backup before I start deleting and updating en mass.
10. db.py
    a. to dedup the database
11. clean_up.py
12. dedup_llm.py
13. irrelevant_rows.py
14. db_maintenance
    a. app.py
        i. Check and make sure it is running fine on Render
    b. main.py
        i. Check and make sure it is running fine on Render.
    c. backup local and psql to render.
        i. Commands are in Security folder.
    d. alter database for changing UTC to PST:
        i. commands are in Security folder
15. cron job
    a. open the editor for cron that lists the jobs to be run. Unfortunately, we will be using nano.

    i. crontab -e

b. this is the command that needs to be put into the file so that it will run
    i. 0 22 */2 * * cd /mnt/d/GitHub/socia_dance_app && /usr/bin/python3 pipelines.py --mode 1 >> /mnt/d/GitHub/socia_dance_app/logs/cron_log.txt 2>&1

URLs Processing Order

NB, we want to do VLDA FIRST because they are copying everybody else's events. So, their events are clobbering everybody else's unless they are processed first. Then their version is killed by the deduplication.

Fixes That Need To Be Done

Urls Table and How We Create, Update, and Use that

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.
    a. 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.
    b. 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.
        i. 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
    a.   # I am pretty sure this is redundant but I will leave it here for now
    b.   if 'login' in url or '/groups/' not in url:
    c.   logging.info(f"def parse(): URL {url} marked as irrelevant due to Facebook login link.")
    d.   update_url(url, update_other_links='No', relevant=False, increment_crawl_trys=0)
    e.   return False

Run Information

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. Only ebs.py and one other have run statistics.
2. You should put the event and url count information that is being kept in output/ events_urls_diff.csv in the database. I think we could just append it to the runs table. We need to instrument the rest of the .pys with run statistics though.

Address Requirements

Thiis is still not working great.

Instructions on Chatbot Page Improved

I just think we need to work on this.

Facebook Scraping

I have gone over this a number of times. I get frustrated with my own code then I get it working, then it does not work. Then I come back and go over the available options. So, lets list them here and the next time (maybe this time) that this happens, I will just use one of them.

Apify – I am concerned about the cost to run this. The per event is not that bad. However, the compute cost seemed ridiculously high. They gave me $5 and it was used up on a trial and I swear I only got 5 events. That did not seem economic, but … it did look like it work.

Bright Data

Marketing

Names

DanceBook

DanceNow

DanceBrain - dancebrain.io

MVP Next Steps

Now that you have a Minimum Viable Product (MVP), the next steps should focus on validating your idea, gathering feedback, and planning for growth. Here are some suggestions:

1. Gather and Analyze User Feedback

- **Beta Testing:** Invite a small group of target users (e.g., local dancers, instructors, and enthusiasts) to test your app. Offer incentives for honest feedback.
- **Surveys and Interviews:** Use surveys or one-on-one interviews to understand what users like and where improvements are needed.
- **Usage Data:** Integrate analytics to monitor how users interact with your app, identifying friction points or features that need refinement.

2. Iterate and Improve

- **Bug Fixes and Enhancements:** Address any issues that emerge from user testing. Prioritize fixes that affect the core functionality of connecting dancers or organizing events.
- **Feature Roadmap:** Identify and plan additional features based on user feedback. For a social dance app, this could include features like event calendars, messaging, group creation, or integration with local dance studios.

3. Strengthen Your Community and Market Position

- **Marketing Strategy:** Develop a marketing plan to attract early adopters. Leverage social media, local dance groups, and partnerships with dance studios.
- **Community Building:** Engage with your audience through content, events, or workshops. Consider hosting local meetups or online sessions that bring the community together.
- **Partnerships:** Reach out to dance instructors, studios, or event organizers to explore collaborations. These partnerships can help validate your app and drive user acquisition.

4. Plan for Scaling

- **Technical Infrastructure:** Ensure your app's backend can handle increased user activity. Plan for scalability if you see rapid growth.
- **Monetization:** Explore monetization options such as premium features, advertising, or event sponsorships. Ensure your revenue strategy aligns with user expectations and community values.
- **Long-term Vision:** Outline a roadmap that balances immediate user needs with future growth and additional features. Prioritize based on user impact and business value.

Taking these steps will help you refine your MVP into a product that not only meets the needs of your users but also stands out in the competitive market of social and event-driven apps.