Ticketmaster Event Data Pipeline

Brandon Habschied | bjh3420@rit.edu

Table of Contents

Section 1 – Introduction

The live event industry is continually evolving [1] through technological integrations that cater to the dynamic needs of eventgoers. Despite the convenience provided by online ticketing platforms like Ticketmaster, StubHub, and SeatGeek, many potential attendees find it challenging to discover events that align precisely with their interests. These platforms often lack user-friendly, intuitive event discovery experiences, leading to potential missed opportunities for both attendees and event organizers.

This capstone aims to address the limitations in browsing for events leveraging the Ticketmaster API to build an efficient data pipeline and a straightforward interface for exploring event data. This capstone focuses on the development of a reliable and scalable system for collecting and organizing event data, laying the groundwork for future applications such as advanced dashboards and analytical tools. These future applications would then be able to redefine how individuals interact with and discover live events, ensuring a more tailored and satisfying experience.

The remainder of this report explores relevant prior work that inspired this project, provides a detailed explanation of the methodology and design choices, outlines obstacles encountered along with their solutions, and concludes with a reflection on the entire design process and future development opportunities.

Section 2 – Prior Work

2.1 Event Discovery Interfaces

This section provides an overview of existing platforms and technologies related to event discovery and ticket sales. It also discusses the broader context of online ticket sales and presents the relevant technologies that have been used in the development of the pipeline.

2.1.1 Ticketmaster

Ticketmaster is one of the most widely used platforms for purchasing tickets to live events. [2] The current interface allows users to search for events using filters such as date, location, and event type. However, the interface lacks advanced personalization options, making it difficult for users to discover events that align with niche preferences. Specifically, Ticketmaster does not allow users to mix and match different preferences into the same search or save preferences for future searches.

2.1.2 StubHub
StubHub offers a similar interface, enabling users to browse and purchase tickets for events worldwide. While it provides similar filters, the platform's event discovery process remains static, relying heavily on users knowing what they are looking for in advance. It has sections for trending events in your area as well as popular categories and last-minute deals for users looking for a quick deal on upcoming events. [3]

2.1.3 SeatGeek
SeatGeek's interface, like the other previously mentioned platforms, lacks in-depth personalization features. SeatGeek specifically lacks the ability to list all events in an area for a date period, regardless of additional filters. [4]

These existing platforms all share the same limitation of being unable to combine different filters when searching for events. If a user is interested in a broad category of events for a specific date, they must decide on which type of events they want to see before they can browse the different available events in the area for the desired date range.

2.2 Relevant Technologies

The development of this capstone project relies on several key technologies to manage the data necessary to address the limitations identified in existing event discovery platforms. These technologies when used create a more dynamic, personalized, and user-friendly interface for event discovery, utilizing real-time data from external sources and providing intuitive, flexible filtering options for users.

2.2.1 Ticketmaster API
The Ticketmaster API serves as the primary data source for retrieving real-time event information, such as event names, locations, dates, and classifications. The API offers various filtering capabilities, allowing developers to search for events based on location, date, and event type. The API enforces a daily rate limit of 5,000 requests and 100 requests per minute, requiring careful management of API calls to stay within the allowed limits while ensuring that enough data is ingested for real-time event exploration. [5]

2.2.2 Snowflake
Snowflake is a cloud-based data warehousing platform that has rapidly grown in popularity due to its flexibility, scalability, and ability to handle diverse data types. Unlike traditional data warehousing solutions, Snowflake operates fully in the cloud, allowing organizations to store and query data without the need for on-premises infrastructure. [6] Users can independently scale storage as data volumes grow, and computing resources can be adjusted dynamically to accommodate querying needs. Snowflake supports structured and semi-structured data formats such as CSV, Parquet, and JSON, making it an ideal choice for handling modern data structures that involve a combination of formats such as found in this capstone. [7]

2.2.3 Streamlit
Streamlit is an open-source application framework designed to simplify the process of building web applications for data science and machine learning applications. [8] Unlike traditional web development frameworks that require a steep learning curve and detailed front-end knowledge, Streamlit allows developers to create fully functional, interactive applications with just a few lines of Python code. [9]

Section 3 – Methodology

This project evolved through an iterative process inspired by agile development principles and techniques, where each step focused on addressing a specific challenge in collecting, processing, storing, and visualizing event data. I approached each task with flexibility and continuous refinement to ensure the system met both immediate needs and minimized long-term tech debt. Broadly, the initial challenge was to begin data ingestion from the Ticketmaster API, then clean and transform the data so it was ready to be stored. Once stored, the data needed to be presented to the dashboard and eventually filtered based on the desired outputs of the end user. During each phase of development, I would review and modify existing code to minimize technical debt or smells that might occur to ease future integration with additional features and to ensure my development was organized and easy to follow. Each of these phases of development is detailed in the following subsections.

3.1 Data Ingestion
The first phase of development, the data ingestion phase, focused on retrieving raw event information from the Ticketmaster API. The first step was to register for an API key and familiarize myself with the API's documentation, particularly its endpoints, parameters, and rate limits. With a limit of 5,000 requests per day and 100 requests per minute, I needed to carefully manage API calls to maximize data retrieval while avoiding unnecessary requests.

The biggest challenge when developing the Data Ingestion with the Ticketmaster API was to overcome the 1000 events per request deep paging limit. Initially, I would only grab the first 1000 events in each day and then move onto the next day. This worked in the beginning of development as it allowed me to have data to explore and develop with, but I knew I would need to change this logic to be able to ingest every event for a given period.

Early solutions to this involved dividing periods of time with more than 1000 events into smaller periods to divide the request into multiple requests. This solution did not successfully consider situations where events were unevenly distributed over a period of time such as a 2-day period having 2000 events but 1900 of those events taking place on the first day and 100 events taking place on the second day. Using my initial logic, I would still miss 900 events on the first day by dividing the API request into 2 separate days.

This phase required several iterations to refine the API request process and error handling. My final solution to this issue was to integrate a recursive function that would continuously divide a request until each divided period of time within the request had less than 1000 events and then process each API request with the new subdivided timestamps. Once I was able to properly ingest all event data for a given period of time, I would take the ingested JSON from the API request and add them to a Pandas Dataframe for future exploration.

3.2 Data Cleaning and Transformation

With the raw event data in a Pandas Dataframe, the phase would have me clean and transform the data before it is stored. To achieve this, I created scripts using Python that ensured the data would be easy to work with and consistent. The responses from the API, formatted in JSON, contained nested structures that included details like event names, locations, dates, and classifications among other information that I wanted to keep in case the information became relevant for later development. The JSON was first written to a Pandas dataframe which would be my preferred storage type for the rest of the project. Some columns within the dataframe included dictionaries or lists of dictionaries which would become problematic for future storage in Snowflake. These nested data structures needed to be flattened to be properly stored. Using a simple for loop and some if statements, I was able to flatten any nested columns using the built in json_normalize function in pandas.

During this phase, I also was able to find that some of the columns had odd incompatible data types that were causing issues when trying to store the dataframe in Snowflake. Due to time constraints and the data being irrelevant or redundant, I did choose to exclude a handful of columns from the final dataframe before loading to Snowflake.

Throughout this phase, when my code would return errors or the dataset was not making sense, I was using a Notebook.ipynb to test the dataset in smaller batches without having to re-request data from the API over and over again. Additionally, before loading any data into Snowflake, I would load the final Dataframe into a CSV file and examine the data there for inconsistencies.

3.3 Data Storage

The next step after having the data cleaned and loaded and transformed into a Pandas Dataframe was to store the data in my Snowflake database. To do this, I first had to create the database in Snowflake and set up all the prerequisites in VSCode to be able to integrate Snowflake into my development stack. The majority of this phase was spent on troubleshooting the dataset when inconsistencies in the data arose from the 2nd phase where the data was not cleaned properly and was returning errors when trying to load into Snowflake. This phase was relatively simple and can be viewed as a validation phase for whether or not the data was successfully cleaned and transformed in the previous phase.

3.4 Dashboard Development

The dashboard development phase focused on presenting the processed event data that is stored in Snowflake in an interactive and user-friendly format using Streamlit. This web-based framework allowed me to quickly build an application with minimal front-end development experience, leveraging its built-in tools for layout, interactivity, and data visualization.

To ensure flexibility and usability, I implemented sidebar controls that allowed users to filter events by various parameters, including date range, location, event type, and price range. These filters were developed as modular components, allowing for future extensions without disrupting existing functionality. The filtered data was displayed in a tabular format, with options for both "Full View" and "Minimal View," giving users the ability to toggle between viewing all the raw data for the filtered events and a more minimalized view of relevant columns with end-user-ready information.

Data for the dashboard was retrieved dynamically from the Snowflake database using SQL queries built based on user-selected filter criteria. To optimize performance, I implemented query caching to stored query results temporarily, reducing redundant database calls and improving response times.

The dashboard provided an additional tab for viewing the entire raw dataset, enabling users and developers to explore all stored event data without filters. This feature served as a reference point for debugging and verifying the integrity of the filtered event information.

Section 4 – Development Challenges and Debugging Strategies

Throughout the development process, as anticipated, I ran into challenges of many diverse types. The earliest of these challenges was setting up the entire VSCode environment for a project of this scale. Creating the Python virtual environment and requirements.txt as well as learning to manage my API keys and other secrets inside a .env file were also early pre-development challenges. An early mistake I made was assuming that I would want the event data to be frequently and automatically updated. This was corrected early in the development process where I transitioned from wanting to frequently pull and replace data to mainly focusing on getting a large amount of data with each request and replacing the entire data set every time I made a new request.

Another development challenge that was encountered was not properly anticipating the need to compartmentalize all my various functions in the early phases of development. I quickly learned that I would be able to be more efficient with a more organized workspace and refactored my code into multiple different scripts that would be referenced from a central main.py script.

As previously discussed, one of the largest challenges of the development process was the 1,000 events per API request limitation and the uneven event distribution across dates. This challenge was overcome through the use of recursion and had multiple phases of troubleshooting. One specific challenge that was encountered multiple times was the inclusion of an accidental infinite loop on my recursive function due to the granularity of the timestamp not being set. This caused me to divide time periods into fractions of a second and continuously create API requests for the same timestamps. The solution to this granularity problem was to make the decision that any timestamps within 10 seconds of each other as non-divisible and to then move onto the next range of timestamps.

When developing the dashboard sidebar filters, there were many challenges when building the dynamic SQL queries based on whether or not one of the filter options was selected or not. The most frequent issue I ran into was when building SQL queries for Snowflake, I found that regardless of the length of the query, using f-strings and single quotes for columns and double quotes for table names was the best way to get consistently successful queries. When initially writing queries with any other combination of normally acceptable quotation formats, there would be frequent errors or lost columns. In the process of overcoming these SQL challenges, I added to the Streamlit application a spot where it would print the SQL query that it built so I could examine it and test it directly in Snowflake or the testing .ipynb.

Section 5 – Future Work

While the project successfully met its primary objectives, there are several areas for future development that could enhance its functionality and availability to end users. Automating the API request process would enable continuous data ingestion over larger date ranges without requiring manual intervention, ensuring the dataset remains up-to-date and comprehensive. The current recursive approach to handling the 1,000-event limit per request could also be optimized by merging smaller API requests into more efficient batches, reducing the overall number of requests and improving performance.

Expanding the data visualizations to include features such as interactive maps, images for events, venues, and artists, and advanced charts would significantly improve the user experience. Although the necessary data is already available, additional front-end development is required to effectively display these features.

One future development I would prioritize given more time would be refactoring the Python-based data validation and transformation processes to integrate popular pipeline frameworks like Apache Airflow or Databricks to significantly enhance scalability and maintainability, while also making the workflow more transparent and easier to monitor and automate.

Future development of the project could incorporate advanced user features, such as the ability to save filters for frequent searches or integrate external APIs like Google Maps or Google Calendars. These enhancements would not only improve the current application but also position it as a more comprehensive and scalable tool for exploring live event data.

Section 6 – Conclusion

This capstone project successfully addressed the challenges of collecting, organizing, and visualizing live event data by leveraging the Ticketmaster API, Snowflake, and Streamlit. The development process emphasized the creation of a reliable and scalable data pipeline capable of managing event data effectively, laying the foundation for future development and applications.

Key takeaways from the project include the successful implementation of a recursive data ingestion process to overcome API limitations, the transformation of raw JSON data into a structured and queryable format, and the design of an interactive dashboard that provides users with a flexible and intuitive interface for event exploration. Each phase of the project demonstrated a commitment to minimizing technical debt and ensuring long-term maintainability, aligning with agile development principles and techniques.

In conclusion, this project demonstrated how modern data tools and techniques can bridge the gap between complex data sources and user-friendly applications, enhancing the way event data is accessed and explored. By addressing the limitations of existing event discovery platforms, this project provides a valuable framework for future developments in improving the live event industry data problem.

Section 7 – References

[1] R. E. Hamilton, "Twenty-First Century Ticketing," Regis University Student Publications, 2006.

[2] "Ticketmaster," [Online]. Available: https://ticketmaster.com/.

[3] "Stubhub," [Online]. Available: https://stubhub.com/.

[4] "Seatgeek," [Online]. Available: https://seatgeek.com/.

[5] "Ticketmaster Discovery API," [Online]. Available: https://developer.ticketmaster.com/products-and-docs/apis/getting-started/.

[6] R. C. ,. B. B. J. ,. B. L. ,. R. S. ,. S. V. Frank Bell, "Snowflake Essentials," 2022.

[7] "CLOUD DATA WAREHOUSING: HOW SNOWFLAKE IS TRANSFORMING BIG DATA MANAGEMENT," 2023.

[8] "Design Recommendations for Intelligent Tutoring Systems," 2020.

[9] M. A. ,. J. H. F. Mohammad Khorasani, "Web Application Development with Streamlit," 2022.