Project Report: Sydney Event Aggregation & Recommendation System

1. Executive Summary

In the bustling city of Sydney, finding the right event can often feel like searching for a needle in a haystack. Current solutions are fragmented, forcing users to jump between ticket sites, social media, and local keys. Our project, the 'Sydney Event Recommendation Assistant', solves this by aggregating data into a single, cohesive ecosystem.

We have built a full-stack automated platform that scrapes events from major sources, cleanses the data, and delivers it through two primary channels: a modern Web Dashboard for browsing and a telegram-based AI Chatbot for personalized, conversational recommendations.

2. The Problem

Users face 'Choice Paralysis' and 'Information Fragmentation'. Data is often outdated, duplicates abound across different platforms, and search filters are rigid (e.g., you can filter by 'Music', but not by 'Vibe'). There was no central intelligence that understood context like 'I want a chill date night in Newtown that costs less than $50'.

3. Technical Architecture

Our solution is built on a robust, modular architecture designed for scalability and autonomy.

Backend (Node.js & Express): The core engine handling API requests and database management. It features an automated Scheduler (Cron) that triggers smart scrapers every hour.

Data Pipeline (MongoDB & ChromaDB): We use a dual-database approach. MongoDB stores the raw, structured event data (dates, venues, prices) for the web frontend. Simultaneously, we process text descriptions into 'Vector Embeddings' using the 'all-MiniLM-L6-v2' model and store them in ChromaDB. This allows our AI to perform 'Semantic Search' - matching concepts rather than just keywords.

Frontend (React & Tailwind CSS): A responsive, high-performance web interface allowing users to filter by date, location, and keywords. It features a dedicated Admin Control Room for curating scraped data before it goes live.

AI Bot (Python & LangChain): The star feature. A Telegram bot that interacts with users naturally. It uses Retrieval-Augmented Generation (RAG) to query our Vector Database and fetch highly relevant events based on user prompts.

4. Challenges & Solutions

Active Data Management: A major challenge was 'zombie events' - events that had passed but were practically still cluttering the DB. We implemented a sophisticated 'Auto-Expire' logic that checks not just the start date, but end dates and recurring schedules to ensure only truly active events are shown.

Dependency Conflicts: Integrating modern AI libraries (LangChain/Ollama) with standard web tools led to version clashes (specifically `httpcore` and `huggingface_hub`). We resolved this by strictly pinning compatible versions in a virtual environment.

5. Future Roadmap

- Personalized User Profiles: Storing long-term user preferences to push proactive notifications.
- Multi-Modal AI: Allowing users to send an image of a flyer to index an event.
- Social Features: 'Group Voting' on events within the Telegram chat.

6. Conclusion

This project bridges the gap between raw data and actionable leisure time. By combining traditional web engineering with modern AI vector search, we've created a tool that feels less like a search engine and more like a knowledgeable local friend.