Event Listing Platform – Full Stack Assignment Report

1. Introduction:

This project is a **full-stack web application** designed to automatically collect, store, and display public events happening in **Sydney, Australia**.
The goal of the assignment is to demonstrate an **end-to-end pipeline** including data scraping, backend APIs, database storage, frontend display, and user interaction.

The system follows a real-world architecture where data is fetched from external sources, processed on the backend, and presented to users through a clean web interface.

2. Project Objectives:

The main objectives of the project are:

- Automatically collect event data from public sources
- Store and manage events in a database
- Display events on a user-friendly website
- Capture user email and consent before redirecting to ticket sources
- Provide a foundation for admin review and future scalability

3. System Architecture:

The project follows a modular architecture with clear separation of concerns.

Components:

1. Backend (Node.js + Express)
2. Database (MongoDB)
3. Frontend (React)
4. Scraper Module (Python)

Data Flow:

Scraper → Backend API → MongoDB → Frontend UI → User Interaction

4. Backend Implementation

4.1 Technologies Used

- Node.js
- Express.js
- MongoDB (Mongoose)
- REST APIs

4.2 Event Management

Events are stored with detailed metadata such as:

- Event title
- City
- Source website
- Original event URL
- Status (new, updated, inactive, imported)
- Timestamps

An **upsert mechanism** is used to:

- Detect new events
- Avoid duplicate entries
- Track updates over time

4.3 Lead Capture

When a user clicks “Get Tickets”:

- Email address and consent are collected
- Data is saved in the database
- User is redirected to the original event website

This ensures **user consent and lead tracking**, aligning with real-world data practices.

---

5. Frontend Implementation

5.1 Technologies Used

- React (Vite)
- Axios
- CSS for styling

5.2 User Interface

The frontend displays:

- A list of Sydney events
- Event cards with title, city, and source
- A “GET TICKETS” call-to-action button

5.3 Get Tickets Flow

Instead of immediate redirection:

1. A modal popup appears
2. User enters email
3. User provides consent
4. Data is saved
5. User is redirected to the original event site

This provides a **clean and professional user experience**.

---

6. Scraper Module

The scraper is implemented as a **separate Python module** to maintain separation from the backend.

Key Points:

- Designed to fetch event data from public sources
- Sends data to the backend using REST APIs
- Does not directly interact with the database
- Can be scheduled using cron jobs

For demo stability, controlled sample data is used to simulate real-world scraping behavior.

---

7. Key Features Demonstrated

- Full-stack MERN-style architecture
- Automated data ingestion pipeline
- RESTful API design
- Database schema design with status tracking
- Frontend–backend integration
- User consent handling
- Clean and modular code structure

---

8. Optional Extension (Future Scope)

The system is designed to be easily extended with:

- Admin dashboard with Google OAuth
- Event review and import workflow
- AI-based event recommendations using open-source LLMs
- User notification system for new matching events

---

9. Conclusion

This project successfully demonstrates an **end-to-end full stack application** that mirrors real-world software systems.
It highlights skills in backend development, frontend integration, database management, and system design.

The modular and scalable architecture ensures that the platform can be enhanced with additional features such as dashboards, analytics, and AI-based recommendations in the future.