**SYDNEY EVENT**

**SCRAPER**

**1. Project Overview**

**Objective**

* **Purpose:**  
  Develop a website that displays all events taking place in Sydney, Australia.
* **Key Features:**
  + **Automated Event Scraping:**  
    Automatically fetch events from a target event website using web scraping.
  + **Beautiful Event Listing:**  
    Display event details (title, date, and booking link) in a user-friendly, responsive layout.
  + **Ticket Request Flow:**  
    On clicking the “GET TICKETS” button, capture the user's email address through an opt-in modal and then redirect them to the original event booking site.
  + **Scheduled Data Updates:**  
    Use a scheduler to update event details automatically every 24 hours.

**2. Project Structure**

The project is divided into two main parts:

* **Backend:**  
  Contains the server-side code responsible for scraping events, providing API endpoints, and handling email submissions.
* **Frontend:**  
  A React-based client application (built with Vite) that fetches event data from the backend and provides an interactive user interface.

**3. Backend Implementation**

**Technology Stack**

* **Language:** Python
* **Framework:** Flask (for REST API development)
* **Libraries:**
  + **Beautiful Soup:** For scraping event data from the source website.
  + **APScheduler:** To schedule the scraping process every 24 hours.
  + **Flask-CORS:** To enable Cross-Origin Resource Sharing (CORS) for API access from the frontend.

**Setup Instructions**

1. **Environment Setup:**  
   Ensure you have Python installed. Then, from the backend folder, install the required dependencies: “pip install -r requirements.txt”.
2. **Running the Server:**  
   To start the Flask server, run: “python server.py”

**Code Explanation: server.py**

* **Imports and Flask Setup:**  
  The script imports Flask, enables CORS, and sets up basic endpoints.
* **Scraping Function (scrape\_events):**
  + **URL Target:** https://allevents.in/sydney/all?ref=new-cityhome-popular
  + **User-Agent Header:** Used to mimic a real browser request.
  + **HTML Parsing:**  
    Uses Beautiful Soup to parse the HTML and locate the container (ul.event-card-parent) holding event cards.
  + **Data Extraction:**  
    Iterates through each <li> element with the class event-card to extract:
    - Event title (from <h3>)
    - Date (from <div class="date">)
    - Booking link (from the data-link attribute)
  + **Error Handling:**  
    Prints an error message if the events container is not found or if another issue occurs.
* **Scheduler Integration:**  
  Uses APScheduler’s BackgroundScheduler to run scrape\_events every 24 hours, ensuring that the events data stays up-to-date.
* **API Endpoints:**
  + **GET /api/events:**  
    Returns the list of scraped events in JSON format.
  + **POST /api/submit-email:**  
    Accepts a JSON payload with the user’s email address, logs the submission, and returns a success message.  
    This endpoint is called when the user clicks the “Continue” button after entering their email.

**Additional File: print\_db.py**

* **Purpose:**  
  This file replicates the functionality of server.py it contains the same scraping logic, scheduler setup, and API endpoints. It may serve for testing, demonstration, or as a backup script.
* **Note:**  
  Although it shares similar functionality with server.py, in a production environment, you would typically have a single, unified backend script.

**4. Frontend Implementation**

**Technology Stack**

* **Framework:** React (using Vite for a fast development environment)
* **Styling:** Bootstrap & React-Bootstrap for responsive and modern UI components.

**Setup Instructions**

1. **Project Initialization:**

npm create vite@latest frontend --template react

cd frontend

npm install

1. **Running the Frontend Server:**

npm run dev

This will launch the development server, and you can access the app (typically at http://localhost:3000 or as indicated in the terminal).

**Code Explanation: App.jsx**

* **Data Fetching:**  
  Uses the useEffect hook to fetch events from the backend endpoint when the component mounts. The fetched data is stored in the state variable events.
* **Event Listing:**
  + Displays events in a grid layout using React-Bootstrap’s Container, Row, and Col components.
  + Each event is shown within a Card component that displays the event title and date.
* **Get Tickets Workflow:**
  + **Button Interaction:**  
    When the user clicks “Get Tickets” on a card, the selected event details are saved, and a modal window is displayed.
  + **Modal Form:**  
    The modal asks the user to enter their email address. This email is captured using a controlled input (useState for managing email state).
  + **Submission Handling:**  
    On clicking “Continue”, the email is sent to the backend via a POST request to /api/submit-email. After a successful response, the user is redirected to the event’s booking link.
* **Styling & Responsiveness:**  
  The combination of Bootstrap classes and custom CSS in App.css ensures that the application is visually appealing and responsive across devices.

**5. Running the Entire Application**

**Steps to Test Locally**

1. **Backend:**
   * Navigate to the backend folder.
   * Install dependencies and run the Flask server (python server.py).  
     The server runs on port 5000 by default.
2. **Frontend:**
   * Navigate to the frontend folder.
   * Install dependencies and start the development server (npm run dev).
3. **User Experience Flow:**
   * Open the frontend application in your browser.
   * Browse through the list of Sydney events.
   * Click “Get Tickets” on an event card to open the email input modal.
   * Submit your email and be redirected to the original event booking page.

**6. Future Enhancements**

* **Database Integration:**  
  Although an app.db file is mentioned, integrating a full-fledged database (e.g., SQLite, PostgreSQL) could allow storing event data persistently and enable features like user account management.
* **Error Handling & Logging:**  
  Improve error handling on both the backend and frontend to provide better user feedback in case of network or data retrieval issues.
* **Security Enhancements:**  
  Implement proper email validation, rate limiting, and other security measures to safeguard the API endpoints.
* **UI/UX Improvements:**  
  Enhance the frontend with animations, better responsiveness, and more detailed event descriptions (e.g., images, locations).
* **Deployment Considerations:**  
  Plan for deployment using platforms such as Heroku, AWS, or DigitalOcean. Consider containerizing the application using Docker for easier deployment and scalability.

**7. Conclusion**

This project successfully creates a web platform that aggregates and displays events in Sydney using automated web scraping, a scheduled update mechanism, and a modern, responsive UI. The backend, built with Flask, handles data scraping and API services, while the frontend, built with React and Bootstrap, offers a seamless user experience.

By following the provided instructions, one can easily set up, run, and further develop the project. The outlined future enhancements offer a pathway for scaling and improving the application, ensuring it remains robust, secure, and user-friendly.

This report encapsulates all necessary details and instructions for your Sydney events website project. If you have any questions or require further details, please feel free to ask!