App Overview

An easy-to-use event recommendation and calendaring system that intelligently suggests sporting events, music concerts, and other events based on user preferences. Focused on Chicago at launch, the app would later be rolled out to other cities.

Front-End

Tech

Front-end development should be done in React Native, as we'd like to have iOS and Android compatibility and the option to have a future web version.

Login & Authentication

The app should have a secure login that uses (ideally multiple) popular authentication services, like Google, Apple, and Meta.

Profile Page

Profile/tastes page that allows the user to select any sports team, band, music venue, or (future) music festivals. The user should be prompted to input their tastes when they first log in to the app (with a “skip for now” option to reduce friction). We may use 3rd party APIs to populate tastes (e.g., Spotify for music). For Spotify, we could do this on a one-time basis upon first API connection with an option to “sync with Spotify” for future updates.

Home Page

The home page should feature a prominent upcoming events calendar, populated with suggested events based on user preferences. Data displayed should include event time, location, and, where possible, a graphical icon to denote the type of event (e.g., team logo for sports events). There should be multiple views of the calendar:

- Day
- Week (default)
- Month

As the calendar zooms out, we can use color-coded shapes or symbols to denote major categories of events falling on that day. More information can be included as we zoom in, including:

- Ticket pricing and links to popular ticketing sites (or this could be on the Event Detail Page)
- (Future) If a sports event, display key betting information, including game lines and over/under, and links to popular betting sites

Below the calendar should be a feed of recommended events. For each event, there should be a graphical element (e.g., a team logo), a title, date, venue, and ticket price (if available). In the future, we may consider displaying a personalized score (see back-end section for more) or an icon to denote that it's highly recommended to that user.

Event Detail Page

Each event should have a detail page clickable from the home page calendar or feed. Similar to the feed on the home page, the detail page should show a graphical element, event title, venue, time, and price. Users should be able to:

- Save/bookmark an event
- Add an event to Google or other calendar
- Share an event via text or social media

Future possibilities include:

- Links to purchase tickets
- Betting information (if sports)
- “Pro tips”: insider information about the event or venue
- Show friends who are attending an event

Future Features & Ideas

- Support for multiple cities and geolocation
- Incorporate bar and restaurant events
- Social networking features
- Tighter Integration with Google Calendar

Design

The app should be dead simple to use, above all. Design should be clean, minimal, and modern.

Back-End

The system would be built around five main components.

1. User Database

Build a database to store user-related data.

- **What it does:** Stores and retrieves user contact information, auth details, tastes, and saved events. Privacy policy TBD.
- **Tech Stack:** PostgreSQL

2. Data Ingestion & Storage ("Event Warehouse")

Build a database or data warehouse of events primarily using 3rd party APIs like Ticketmaster, Stubhub, Seatgeek (we will start with only one of these services to start), Eventbrite, Spotify, Balldontlie.io (sports teams and scheduling).

- **What it does:** Runs automated, scheduled tasks to fetch event data from API sources (Ticketmaster, Stubhub, Eventbrite, etc.) for supported cities. This system also performs a de-duplication process to ensure the same event isn’t repeated (e.g., Stubhub shows ‘Spoon’ and Ticketmaster shows ‘Spoon w/special guests’).
- **Tech Stack:**

        - **Job/Message Queue (e.g., `Celery`, `Sidekiq`, `RabbitMQ`):** Put "jobs" (like "Fetch Chicago Events") into a queue, and background "workers" complete them.
        - **Scheduler (e.g., `Cron`, `AWS Lambda Scheduled Events`):** A timer that triggers these jobs. For example: "Every 4 hours, fetch all events in Chicago for the next 90 days."
        - **Data Storage (Primary):** A database to store the raw event data.
            - **Relational DB (e.g., `PostgreSQL`):** This is the favored approach given that we'd like to minimize the number of different technologies used.
            - **NoSQL DB (e.g., `MongoDB`)**

3. Search & Filtering Engine

Create a way to quickly filter and search the database for relevant events.

- **What it does:** Takes thousands of events and makes them instantly searchable by any combination of factors (date, location, category, price, artist).
- **Tech Stack:**
    - **Search Engine (e.g., `Elasticsearch` or `Algolia`):** All events are indexed here. The app's query for "Chicago this weekend" would hit Elasticsearch, not the main database.

4. Recommendation Engine

This is the logic that decides *what* to show the user. It starts simple and gets more complex.

- **Phase 1: Rule-Based / Content-Based Filtering (Your "Must-Have")**
    - **What it is:** This is an "intelligent filter" based on a set of rules:
        - `IF` user follows "Spoon" on Spotify, `THEN` show all "Spoon" events.
        - `IF` user follows "Chicago Bears" in the survey, `THEN` show all "Chicago Bears" events.
    - **Tech:** This is simply a smart query to the Search Engine (Elasticsearch).
- **Phase 2: Collaborative Filtering**
    - **What it is:** The "people like you also liked..." logic.
        - "Users who like Spoon *also* like The National."
        - "Users who go to Chicago Bears games *also* go to Chicago Bulls games."
    - **Tech:** Machine learning with **Python** using libraries like `scikit-learn` or `Surprise` to analyze user behavior (what they *actually* click on and save). Or use managed cloud services like **Amazon Personalize** or **Google Cloud Recommendations AI** to handle this.
- **Phase 3: The Ranking Model**
    - **What it is:** This answers "what are the *top* events?" Devise a scoring algorithm to show a personalized subset of events ranked by score.
    - **Tech:** This is a custom algorithm that assigns points to events and sorts in descending order:
        - Direct artist match (Spoon): +100 points
        - Related artist (The National): +50 points

        - Event is very popular: +20 points
        - Event is tomorrow: +15 points
        - Event is 3 months away: +5 points

5. Application API

This is the backend server that the mobile app actually talks to.

- **What it does:** The user's app makes a simple request like `GET /recommendations`. This API then does all the work: gets the user's profile, queries the recommendation engine, and returns a clean, simple JSON list of the top X events.
- **Tech Stack:**
    - **Backend Framework (e.g., `Node.js`/`Express`, `Python`/`FastAPI`/`Django`, `Ruby on Rails`):** The code that handles the web requests and runs all the logic.