**Step 1: Define Requirements**

* **Features**:
  + Users can select a day, genre, and location.
  + API returns a list of nearby events.
  + An interactive map displays event locations.
  + Optional: Include filtering by event type (e.g., movie, panel, workshop).
* **Technology Stack**:
  + Backend: Flask/Django (Python) or Node.js.
  + Database: PostgreSQL with PostGIS (for geospatial queries) or MongoDB.
  + Mapping API: Leaflet.js, Google Maps API, or OpenStreetMap.
  + Deployment: Heroku, AWS, or Vercel.

**Step 2: Design Database**

* **Tables/Collections**:
  1. **Events Table**:
     + id: Unique event identifier.
     + name: Event name.
     + description: Event details.
     + date: Date and time of the event.
     + genre: Genre or type of event (e.g., drama, workshop).
     + location: Geolocation coordinates (latitude, longitude).
     + venue: Address or venue name.
  2. **Venues Table** (optional):
     + id: Venue identifier.
     + name: Venue name.
     + address: Full address.
     + coordinates: Latitude and longitude.

**Step 3: Implement API Endpoints**

* **API Structure**:
  1. GET /events: Returns a list of events based on filters (date, genre, location).
     + **Parameters**:
       - date (optional): Filter events by day.
       - genre (optional): Filter events by genre.
       - location (required): Latitude, longitude of the user’s location.
       - radius (optional): Distance in kilometers/miles.
     + **Response**: List of events with geolocation details.
  2. GET /event/{id}: Returns detailed information about a specific event.
     + **Parameters**: id (event ID).

**Step 4: Build Interactive Map**

* Use a frontend framework (React, Vue.js, or plain JavaScript).
* Integrate a mapping library:
  + **Markers**: Display events as clickable markers.
  + **Popups**: Show event details when a user clicks a marker.
  + **Search Box**: Allow users to input their location or select filters.

**Step 5: Geospatial Queries**

* Use geospatial capabilities for filtering events by proximity:
  + PostgreSQL: ST\_DWithin for calculating distance between user location and event venues.
  + MongoDB: $geoNear for similar geospatial queries.

**Step 6: Notifications**

* Add optional user notifications:
  + Store user preferences for events they want to follow.
  + Send email or app notifications using tools like Twilio SendGrid or Firebase.

**Step 7: Testing**

* Write unit tests for each API endpoint using tools like Postman or Pytest.
* Test map interactivity and geospatial filtering.

**Step 8: Deployment**

* Host the API and frontend:
  + Backend: Deploy on Heroku, AWS, or Render.
  + Frontend: Host on Netlify or Vercel.
  + Mapping API: Ensure appropriate API keys and quotas for public use.

**Bonus Features (Optional Enhancements):**

* Add support for event reviews or ratings.
* Allow users to bookmark or share events.
* Show route suggestions to venues using mapping APIs.

**Blueprint Files in resources/ Folder**

**1. movies.py**

Handles CRUD operations for the movies table.

**Endpoints:**

* GET /movies: Retrieve all movies.
* GET /movies/<movie\_id>: Retrieve a specific movie by its ID.
* POST /movies: Add a new movie.
* PUT /movies/<movie\_id>: Update an existing movie by ID.
* DELETE /movies/<movie\_id>: Delete a movie by ID.

**2. programs.py**

Handles CRUD operations for the programs table.

**Endpoints:**

* GET /programs: Retrieve all programs.
* GET /programs/<program\_id>: Retrieve a specific program by ID.
* POST /programs: Add a new program.
* PUT /programs/<program\_id>: Update an existing program by ID.
* DELETE /programs/<program\_id>: Delete a program by ID.

**3. venues.py**

Handles CRUD operations for the venues table.

**Endpoints:**

* GET /venues: Retrieve all venues.
* GET /venues/<venue\_id>: Retrieve a specific venue by ID.
* POST /venues: Add a new venue.
* PUT /venues/<venue\_id>: Update an existing venue by ID.
* DELETE /venues/<venue\_id>: Delete a venue by ID.

**4. agenda.py**

Handles CRUD operations for the agenda table.

**Endpoints:**

* GET /agenda: Retrieve all agenda entries.
* GET /agenda/<agenda\_id>: Retrieve a specific agenda entry by ID.
* POST /agenda: Add a new agenda entry.
* PATCH/agenda/<agenda\_id>: Update an existing agenda entry by ID.
* DELETE /agenda/<agenda\_id>: Delete an agenda entry by ID.
* GET /agenda/venue/<venue\_id>: Retrieve agenda entries for a specific venue.
* GET /agenda/movie/<movie\_id>: Retrieve agenda entries for a specific movie.

**Examples of Additional Endpoints**

**Custom Endpoint Ideas**

1. **GET /programs/<program\_id>/movies**
   * Retrieve all movies in a specific program.
2. **GET /venues/<venue\_id>/agenda**
   * Retrieve all agenda entries for a specific venue.
3. **GET /movies/<movie\_id>/agenda**
   * Retrieve all agenda entries for a specific movie.
4. **GET /movies/search?title=<title>**
   * Search for movies by title.
5. **GET /venues/type/<type>**
   * Retrieve venues filtered by type (cinema, theatre, etc.).