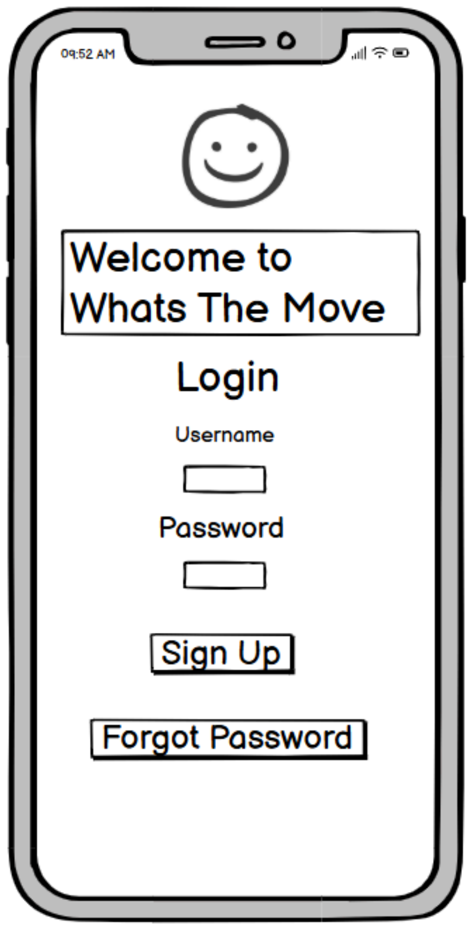
What's the Move - Design Document

Table of Contents

UI Design

For the UI design we chose 6 use cases, below we will share the wireframes demonstrating the UI and add context to each wireframe.

Use Case One:



Inputs:

- Email or Username field.
- Password field.
- "Sign up" link/button for new users.

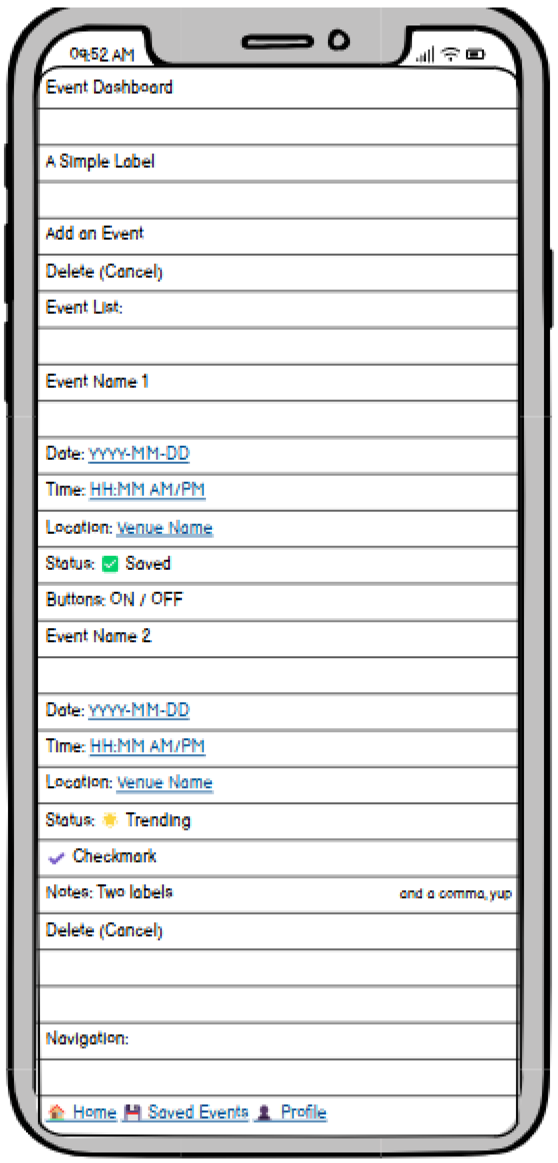
Outputs:

- Error messages (e.g., "Invalid login credentials").

Actions:

- "Login" button to proceed.
- "Forgot Password?" link.

Use Case Two:


09:52 AM

Event Dashboard

A Simple Label

Add an Event

Delete (Cancel)

Event List:

Event Name 1

Date: YYYY-MM-DD

Time: HH:MM AM/PM

Location: Venue Name

Status: ✓ Saved

Buttons: ON / OFF

Event Name 2

Date: YYYY-MM-DD

Time: HH:MM AM/PM

Location: Venue Name

Status: ✹ Trending

✓ Checkmark

Notes: Two labels and a comma, yup

Delete (Cancel)

Navigation:

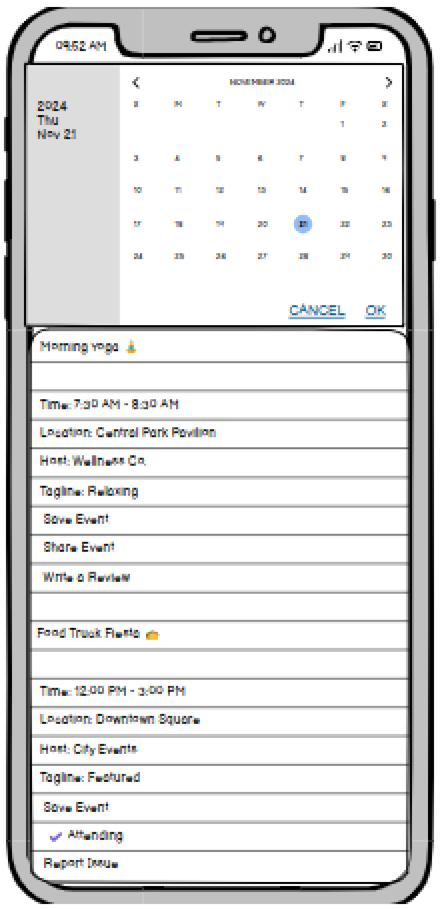
Home Saved Events Profile

Purpose: Displays all nearby events for the user to browse.

Elements:

- **Outputs**:
  - List of events (e.g., title, date, time, category).
  - Filters (e.g., by location, category, or popularity).
- **Actions**:
  - "Save Event" button next to each event.
  - Search bar for events by keyword.
  - Navigation bar (e.g., Home, My Events, Notifications).

Use Case Three:



Morning yoga 🧘

Time: 7:30 AM - 8:30 AM

Location: Central Park Pavilion

Host: Wellness Co.

Tagline: Relaxing

Save Event

Share Event

Write a Review

Food Truck Fiesta 🌮

Time: 12:00 PM - 3:00 PM

Location: Downtown Square

Host: City Events

Tagline: Featured

Save Event

✓ Attending

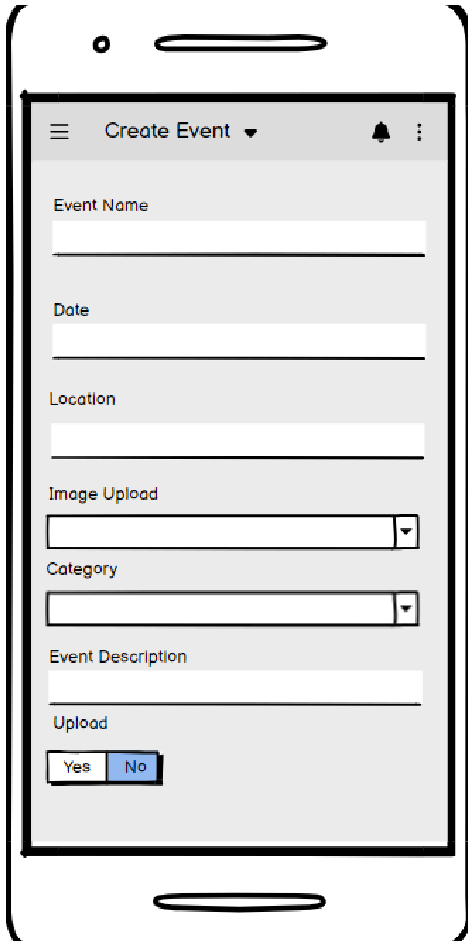
Report Issue

Purpose: Shows detailed information about a specific event.

Elements:

- **Outputs:**
  - **Event title, date, time, description, venue info.**
  - **Reviews (e.g., user ratings, comments).**
- **Actions:**
  - **"Save Event" button.**
  - **"Write a Review" button.**
  - **"Share Event" button**

Use Case Four:

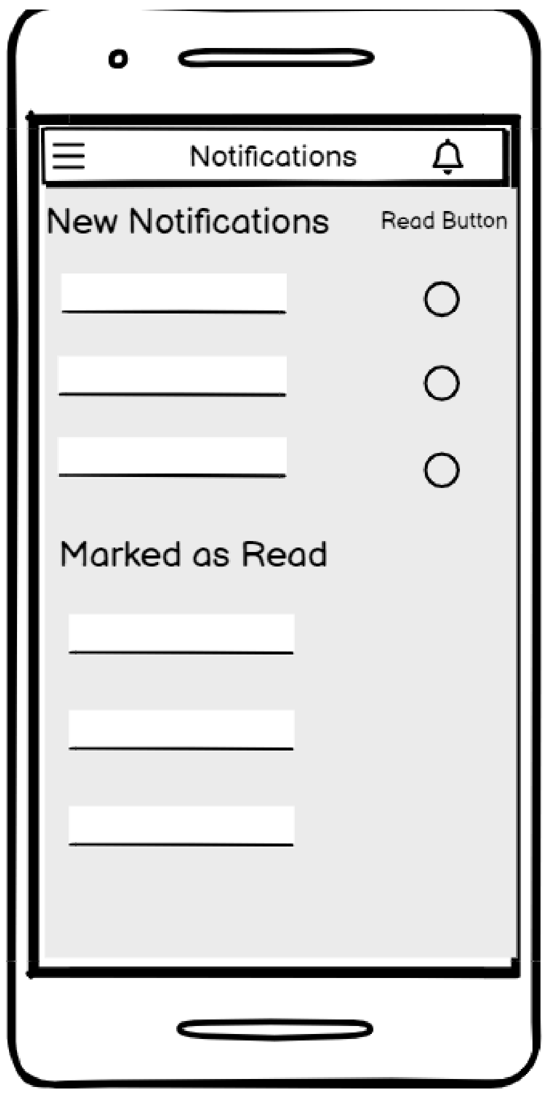


Purpose: Allows venue users to create or modify events.

Elements:

- **Inputs:**
  - Event title, description, date, time, and category fields.
  - Location/venue field.
  - Image upload option.
- **Actions:**
  - "Save" or "Publish Event" button.
  - "Cancel" button.

Use Case Five



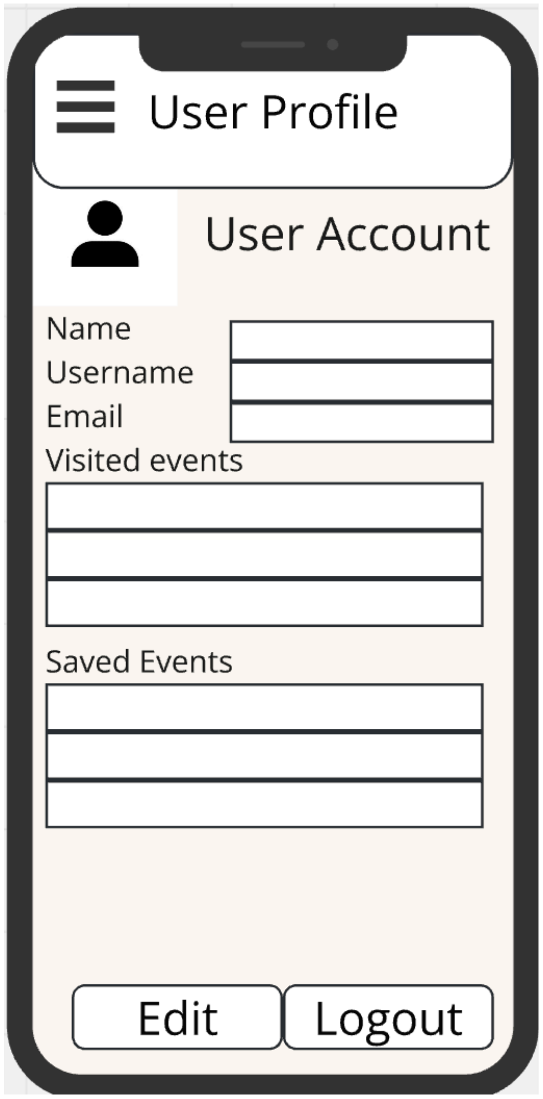
Notifications Screen

Purpose: Lists updates about saved events, new event postings, or reminders.

Elements:

- **Outputs:**
  - **Notification list with timestamps.**
  - **Clear visual indicators for new notifications.**
- **Actions:**
  - **"Mark as Read" option for each notification.**
  - **Navigation bar to return to Dashboard.**

Use Case Six



User Profile Screen

Purpose: Allows users to view and manage their account and saved events.

Elements:

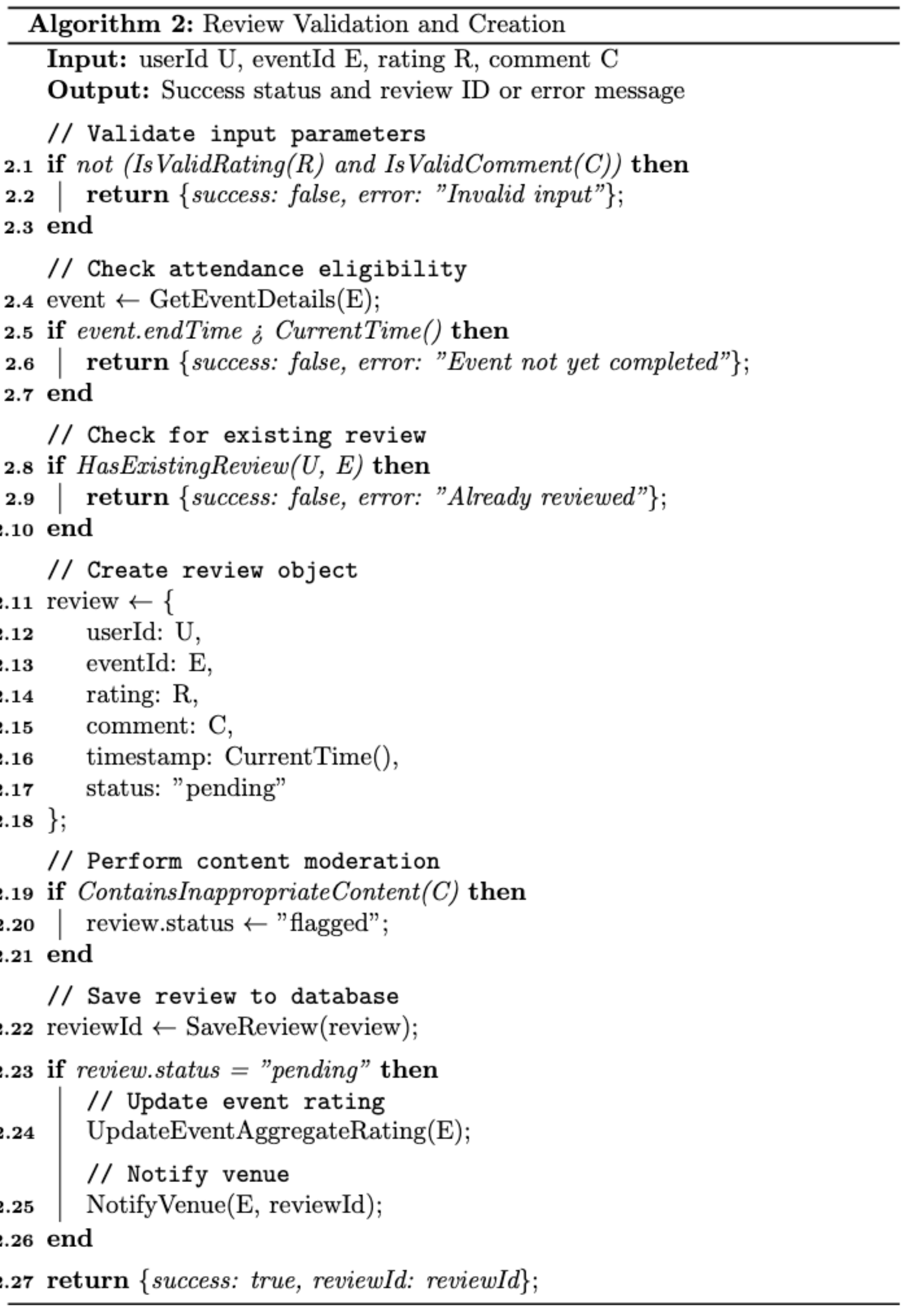
- **Outputs:**
    - **List of saved events.**
    - **Reviews written by the user.**
- **Actions:**
    - **"Edit Profile" button.**
    - **"Logout" button.**

Algorithm Design

Algorithm 1: Event Discovery Algorithm

```
Algorithm 1: Event Discovery Algorithm
Input: location L, userId U, filters F (optional)
Output: Sorted list of relevant events E
Initialize empty event list E;
userPrefs ← GetUserPreferences(U);
radius ← DEFAULT_RADIUS;
if F contains custom_radius then
    radius ← F.radius;
end
foreach category C in F.categories do
    queryStr ← BuildCategoryQuery(C);
    events ← QueryDatabase(queryStr, L, radius);
    foreach event e in events do
        score ← 0;
        // Calculate distance score
        dist ← CalculateDistance(L, e.location);
        score += (radius - dist) / radius * DISTANCE_WEIGHT;
        // Calculate time relevance
        if IsUpcoming(e.startTime) then
            score += TIME_WEIGHT;
        end
        // Calculate preference match
        if e.category in userPrefs.favoriteCategories then
            score += PREFERENCE_WEIGHT;
        end
        // Add to results if meets threshold
        if score ≥ RELEVANCE_THRESHOLD then
            E.append({event: e, score: score});
        end
    end
end
SortByScore(E);
return E.take(MAX_RESULTS);
```

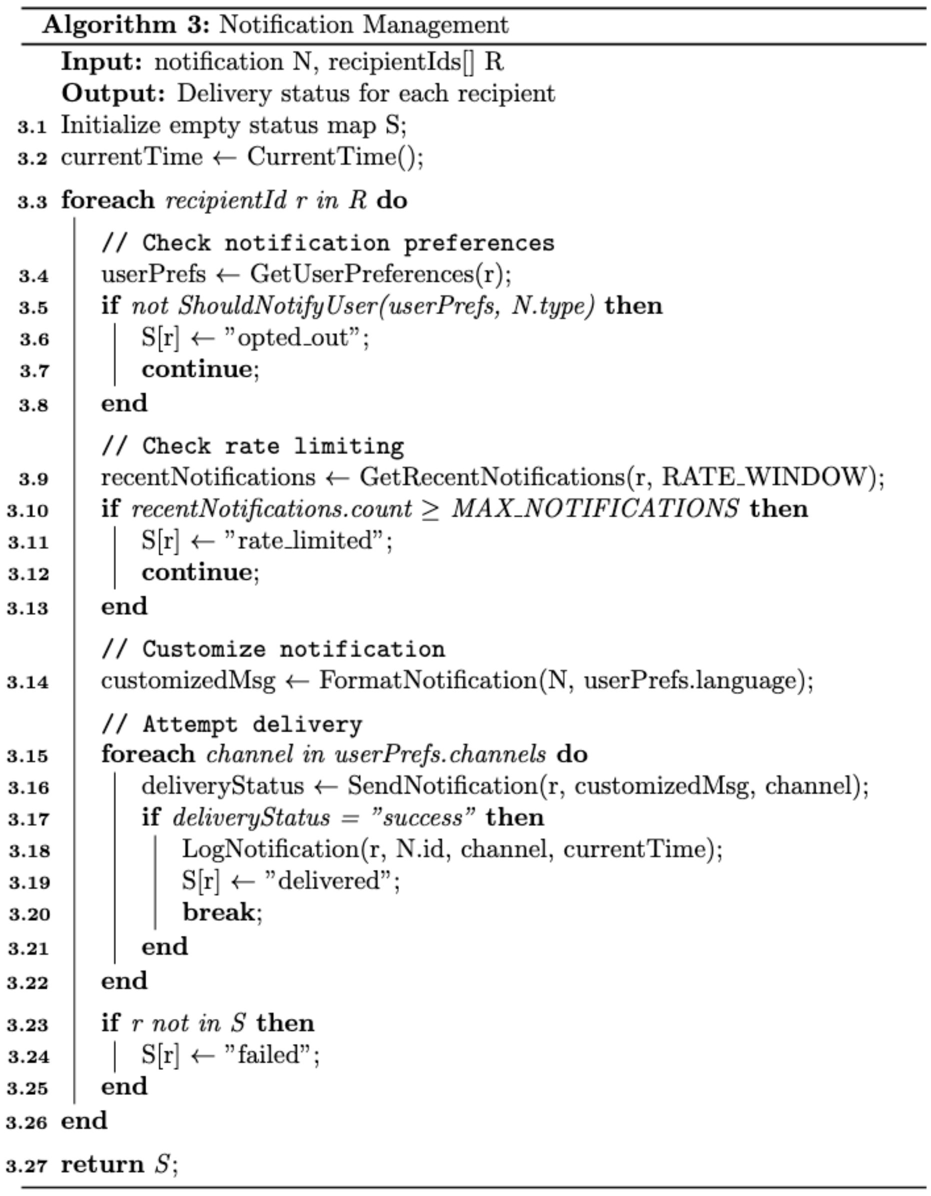
Algorithm 2: Review Validation and Creation



Algorithm 2: Review Validation and Creation

```
Input: userId U, eventId E, rating R, comment C
Output: Success status and review ID or error message
// Validate input parameters
if not (IsValidRating(R) and IsValidComment(C)) then
    return {success: false, error: "Invalid input"};
end
// Check attendance eligibility
event ← GetEventDetails(E);
if event.endTime ¿ CurrentTime() then
    return {success: false, error: "Event not yet completed"};
end
// Check for existing review
if HasExistingReview(U, E) then
    return {success: false, error: "Already reviewed"};
end
// Create review object
review ← {
    userId: U,
    eventId: E,
    rating: R,
    comment: C,
    timestamp: CurrentTime(),
    status: "pending"
};
// Perform content moderation
if ContainsInappropriateContent(C) then
    review.status ← "flagged";
end
// Save review to database
reviewId ← SaveReview(review);
if review.status = "pending" then
    // Update event rating
    UpdateEventAggregateRating(E);
    // Notify venue
    NotifyVenue(E, reviewId);
end
return {success: true, reviewId: reviewId};
```

Algorithm 3: Notification Management



Algorithm 3: Notification Management

```
     Input: notification N, recipientIds[] R
     Output: Delivery status for each recipient
3.1  Initialize empty status map S;
3.2  currentTime ← CurrentTime();
3.3  foreach recipientId r in R do
         // Check notification preferences
3.4      userPrefs ← GetUserPreferences(r);
3.5      if not ShouldNotifyUser(userPrefs, N.type) then
3.6          S[r] ← "opted_out";
3.7          continue;
3.8      end
         // Check rate limiting
3.9      recentNotifications ← GetRecentNotifications(r, RATE_WINDOW);
3.10     if recentNotifications.count ≥ MAX_NOTIFICATIONS then
3.11         S[r] ← "rate_limited";
3.12         continue;
3.13     end
         // Customize notification
3.14     customizedMsg ← FormatNotification(N, userPrefs.language);
         // Attempt delivery
3.15     foreach channel in userPrefs.channels do
3.16         deliveryStatus ← SendNotification(r, customizedMsg, channel);
3.17         if deliveryStatus = "success" then
3.18             LogNotification(r, N.id, channel, currentTime);
3.19             S[r] ← "delivered";
3.20             break;
3.21         end
3.22     end
3.23     if r not in S then
3.24         S[r] ← "failed";
3.25     end
3.26 end
3.27 return S;
```

Algorithm 4: Event Creation and Validation

Algorithm 4: Event Creation and Validation

```
Input: venueId V, eventDetails D
Output: Created event ID or error message
// Validate venue authorization
if not IsAuthorizedVenue(V) then
    return {error: "Unauthorized venue"};
end
// Validate event timing
if not IsValidEventTiming(D.startTime, D.endTime) then
    return {error: "Invalid event timing"};
end
// Validate venue capacity
venue ← GetVenueDetails(V);
existingEvents ← GetOverlappingEvents(V, D.startTime, D.endTime);
if existingEvents.count $\geq$ venue.maxSimultaneousEvents then
    return {error: "Venue scheduling conflict"};
end
// Create event object
event ← {
    venueId: V,
    name: D.name,
    description: D.description,
    category: D.category,
    startTime: D.startTime,
    endTime: D.endTime,
    status: "scheduled",
    createdAt: CurrentTime()
};
// Validate category
if not IsValidCategory(event.category) then
    return {error: "Invalid category"};
end
// Save event
eventId ← SaveEvent(event);
// Initialize analytics
CreateEventAnalytics(eventId);
// Notify subscribers
NotifyVenueSubscribers(V, eventId);
return {success: true, eventId: eventId};
```