// Show toast notification
errorHandler.showToast('Link copied to clipboard!', 'success');
}

```
  function showError(message) {
    // Hide loading and details
    document.getElementById('event-loading').classList.add('hidden');
    document.getElementById('event-details').classList.add('hidden');

    // Show error
    const errorElement = document.getElementById('event-error');
    errorElement.classList.remove('hidden');

    console.error(message);
  }
```

</script> </body> </html> ```

2.2. Create RSVP Modal (Duration: 1 day)

```
html
```

```
<!-- Create file: components/modals/rsvp-modal.html -->
<div class="fixed inset-0 bg-black bg-opacity-50 hidden flex items-center justify-center z-50"
  <div class="bg-white rounded-xl w-11/12 max-w-md p-6" onclick="event.stopPropagation();">
    <div class="flex justify-between items-center mb-4">
      <h3 class="text-xl font-bold">RSVP Confirmation</h3>
      <button class="text-gray-500 hover:text-gray-700 close-modal-btn">
        <i class="fas fa-times"></i>
      </button>
    </div>

    <div class="mb-6 text-center">
      <i class="fas fa-calendar-check text-5xl text-purple-600 mb-4"></i>
      <h4 class="text-lg font-semibold event-title">Event Title</h4>
      <p class="text-gray-600 event-date mt-1">Event Date</p>
      <p class="text-gray-600 event-location mt-1">Event Location</p>
    </div>

    <div class="mb-6">
      <p class="text-gray-700 mb-4">Would you like to receive a reminder notification for this
      <div class="flex items-center mb-3">
        <input type="radio" id="reminder-day" name="reminder" value="day" class="mr-2" checked>
        <label for="reminder-day">1 day before</label>
      </div>
      <div class="flex items-center mb-3">
        <input type="radio" id="reminder-hour" name="reminder" value="hour" class="mr-2">
        <label for="reminder-hour">1 hour before</label>
      </div>
      <div class="flex items-center">
        <input type="radio" id="reminder-none" name="reminder" value="none" class="mr-2">
        <label for="reminder-none">No reminder</label>
      </div>
    </div>

    <div class="flex justify-end space-x-3">
      <button type="button" class="px-6 py-3 rounded-lg border border-gray-300 text-gray-700 ho
      <button type="button" id="confirm-rsvp" class="btn-primary px-6 py-3">Confirm RSVP</buttc
    </div>
  </div>
</div>

<script>
  // Close modal functionality
  document.querySelectorAll('#rsvpModal .close-modal-btn').forEach(btn => {
```

```
  btn.addEventListener('click', function() {
    const modal = document.getElementById('rsvpModal');
    modal.classList.add('hidden');
    document.body.style.overflow = 'auto';
  });
});

// Close when clicking outside
document.getElementById('rsvpModal').addEventListener('click', function(e) {
  if (e.target === this) {
    this.classList.add('hidden');
    document.body.style.overflow = 'auto';
  }
});

// Confirm RSVP
document.getElementById('confirm-rsvp').addEventListener('click', async function() {
  // Get the event ID from the modal
  const modal = document.getElementById('rsvpModal');
  const eventId = modal.getAttribute('data-event-id');

  if (!eventId) {
    console.error('No event ID found');
    return;
  }

  // Get selected reminder option
  const reminderOption = document.querySelector('input[name="reminder"]:checked').value;

  // Import needed modules
  const errorHandler = (await import('../../scripts/utils/error-handler.js')).default;

  // Disable the button while processing
  this.disabled = true;
  this.innerHTML = '<i class="fas fa-spinner fa-spin mr-2"></i> Processing...';

  try {
    // Get current user
    const user = firebase.auth().currentUser;
    if (!user) {
      errorHandler.showToast('You must be logged in to RSVP', 'error');
      closeModal();
      return;
    }
```

```
    // Get event data
    const db = firebase.firestore();
    const eventDoc = await db.collection('events').doc(eventId).get();

    if (!eventDoc.exists) {
      errorHandler.showToast('Event not found', 'error');
      closeModal();
      return;
    }

    const eventData = eventDoc.data();

    // Process the RSVP
    const attendeeData = {
      userId: user.uid,
      eventId: eventId,
      eventTitle: eventData.title,
      createdAt: new Date(),
      eventDate: eventData.dateTime || new Date(),
      reminderSetting: reminderOption,
      reminderSent: false
    };

    // Add to attendees collection
    await db.collection('attendees').add(attendeeData);

    // Increment event attendee count
    await db.collection('events').doc(eventId).update({
      attendees: firebase.firestore.FieldValue.increment(1)
    });

    // Close the modal
    closeModal();

    // Update the UI - this will refresh the event detail page UI
    window.location.reload();

    // Show success message
    errorHandler.showToast('You\'re all set! Your RSVP has been confirmed.', 'success');
  } catch (error) {
    console.error('RSVP error:', error);
    errorHandler.showToast('Failed to process RSVP. Please try again.', 'error');
```

```
        // Reset button
        this.disabled = false;
        this.innerHTML = 'Confirm RSVP';
      }
    });

    function closeModal() {
      const modal = document.getElementById('rsvpModal');
      modal.classList.add('hidden');
      document.body.style.overflow = 'auto';

      // Reset button
      const confirmBtn = document.getElementById('confirm-rsvp');
      confirmBtn.disabled = false;
      confirmBtn.innerHTML = 'Confirm RSVP';
    }
  </script>
```

2.3. Create Event Service Module (Duration: 2 days)

```
javascript
```

```
// Create file: scripts/services/event-service.js
import errorHandler from '../utils/error-handler.js';

class EventService {
  constructor() {
    this.db = firebase.firestore();
    this.storage = firebase.storage();
  }

  async getEventById(eventId) {
    try {
      const doc = await this.db.collection('events').doc(eventId).get();

      if (!doc.exists) {
        throw new Error('Event not found');
      }

      return {
        id: doc.id,
        ...doc.data()
      };
    } catch (error) {
      console.error('Error fetching event:', error);
      throw error;
    }
  }

  async getEvents(filters = {}, limit = 10) {
    try {
      let query = this.db.collection('events');

      // Apply filters
      if (filters.category) {
        query = query.where('category', '==', filters.category);
      }

      if (filters.dateRange) {
        const { start, end } = filters.dateRange;
        if (start) {
          query = query.where('dateTime', '>=', start);
        }
        if (end) {
          query = query.where('dateTime', '<=', end);
```

```
      }
    }

    // Apply sorting
    query = query.orderBy(filters.orderBy || 'dateTime', filters.orderDirection || 'asc');

    // Apply limit
    query = query.limit(limit);

    const snapshot = await query.get();

    return snapshot.docs.map(doc => ({
      id: doc.id,
      ...doc.data()
    }));
  } catch (error) {
    console.error('Error fetching events:', error);
    throw error;
  }
}

async searchEvents(searchTerm, limit = 10) {
  try {
    // Basic search implementation - will be enhanced in the future
    const snapshot = await this.db.collection('events')
      .orderBy('title')
      .startAt(searchTerm)
      .endAt(searchTerm + '\uf8ff')
      .limit(limit)
      .get();

    return snapshot.docs.map(doc => ({
      id: doc.id,
      ...doc.data()
    }));
  } catch (error) {
    console.error('Error searching events:', error);
    throw error;
  }
}

async createEvent(eventData, imageFile = null) {
  const user = firebase.auth().currentUser;
  if (!user) {
```

```
    throw new Error('User must be logged in to create an event');
  }

  try {
    // Upload image if provided
    let imageUrl = null;
    if (imageFile) {
      imageUrl = await this.uploadEventImage(imageFile);
    }

    // Create event document
    const eventDoc = {
      ...eventData,
      createdBy: user.uid,
      createdAt: new Date(),
      updatedAt: new Date(),
      attendees: 0,
      imageUrl: imageUrl
    };

    // Add to Firestore
    const docRef = await this.db.collection('events').add(eventDoc);

    return {
      id: docRef.id,
      ...eventDoc
    };
  } catch (error) {
    console.error('Error creating event:', error);
    throw error;
  }
}

async updateEvent(eventId, eventData, imageFile = null) {
  const user = firebase.auth().currentUser;
  if (!user) {
    throw new Error('User must be logged in to update an event');
  }

  try {
    // Check if user is the creator of the event
    const eventDoc = await this.db.collection('events').doc(eventId).get();

    if (!eventDoc.exists) {
```

```
      throw new Error('Event not found');
    }

    const eventCreator = eventDoc.data().createdBy;

    if (eventCreator !== user.uid) {
      throw new Error('You do not have permission to edit this event');
    }

    // Upload image if provided
    let imageUrl = eventDoc.data().imageUrl;
    if (imageFile) {
      imageUrl = await this.uploadEventImage(imageFile);
    }

    // Update event document
    const updateData = {
      ...eventData,
      updatedAt: new Date(),
      imageUrl: imageUrl
    };

    // Update in Firestore
    await this.db.collection('events').doc(eventId).update(updateData);

    return {
      id: eventId,
      ...updateData
    };
  } catch (error) {
    console.error('Error updating event:', error);
    throw error;
  }
}

async deleteEvent(eventId) {
  const user = firebase.auth().currentUser;
  if (!user) {
    throw new Error('User must be logged in to delete an event');
  }

  try {
    // Check if user is the creator of the event
    const eventDoc = await this.db.collection('events').doc(eventId).get();
```

```
    if (!eventDoc.exists) {
      throw new Error('Event not found');
    }

    const eventCreator = eventDoc.data().createdBy;

    if (eventCreator !== user.uid) {
      throw new Error('You do not have permission to delete this event');
    }

    // Delete the event
    await this.db.collection('events').doc(eventId).delete();

    // TODO: Also delete related data (RSVPs, vibe checks, etc.)

    return true;
  } catch (error) {
    console.error('Error deleting event:', error);
    throw error;
  }
}

async uploadEventImage(file) {
  try {
    // Create a storage reference
    const storageRef = this.storage.ref();
    const fileRef = storageRef.child(`event-images/${Date.now()}_${file.name}`);

    // Upload the file
    const snapshot = await fileRef.put(file);

    // Get the download URL
    const downloadURL = await snapshot.ref.getDownloadURL();

    return downloadURL;
  } catch (error) {
    console.error('Error uploading image:', error);
    throw error;
  }
}

async toggleBookmark(eventId) {
  const user = firebase.auth().currentUser;
```

```
  if (!user) {
    throw new Error('User must be logged in to bookmark events');
  }

  try {
    // Check if the event is already bookmarked
    const snapshot = await this.db.collection('bookmarks')
      .where('userId', '==', user.uid)
      .where('eventId', '==', eventId)
      .get();

    if (snapshot.empty) {
      // Add bookmark
      await this.db.collection('bookmarks').add({
        userId: user.uid,
        eventId: eventId,
        createdAt: new Date()
      });

      return true; // Bookmark added
    } else {
      // Remove bookmark
      const batch = this.db.batch();

      snapshot.forEach(doc => {
        batch.delete(doc.ref);
      });

      await batch.commit();

      return false; // Bookmark removed
    }
  } catch (error) {
    console.error('Error toggling bookmark:', error);
    throw error;
  }
}

async isBookmarked(eventId) {
  const user = firebase.auth().currentUser;
  if (!user) return false;

  try {
    const snapshot = await this.db.collection('bookmarks')
```

```
      .where('userId', '==', user.uid)
      .where('eventId', '==', eventId)
      .get();

    return !snapshot.empty;
  } catch (error) {
    console.error('Error checking bookmark status:', error);
    return false;
  }
}

async getBookmarkedEvents() {
  const user = firebase.auth().currentUser;
  if (!user) return [];

  try {
    const snapshot = await this.db.collection('bookmarks')
      .where('userId', '==', user.uid)
      .get();

    const bookmarkIds = snapshot.docs.map(doc => doc.data().eventId);

    if (bookmarkIds.length === 0) return [];

    // Get the actual events
    const eventsSnapshot = await this.db.collection('events')
      .where(firebase.firestore.FieldPath.documentId(), 'in', bookmarkIds)
      .get();

    return eventsSnapshot.docs.map(doc => ({
      id: doc.id,
      ...doc.data()
    }));
  } catch (error) {
    console.error('Error getting bookmarked events:', error);
    return [];
  }
}

async rsvpToEvent(eventId, reminderSetting = 'day') {
  const user = firebase.auth().currentUser;
  if (!user) {
    throw new Error('User must be logged in to RSVP');
  }
```

```
try {
  // Check if already RSVP'd
  const attendeeSnapshot = await this.db.collection('attendees')
    .where('userId', '==', user.uid)
    .where('eventId', '==', eventId)
    .get();

  if (!attendeeSnapshot.empty) {
    throw new Error('You have already RSVP\'d to this event');
  }

  // Get event details
  const eventDoc = await this.db.collection('events').doc(eventId).get();

  if (!eventDoc.exists) {
    throw new Error('Event not found');
  }

  const eventData = eventDoc.data();

  // Begin a batch write
  const batch = this.db.batch();

  // Add attendee record
  const attendeeRef = this.db.collection('attendees').doc();
  batch.set(attendeeRef, {
    userId: user.uid,
    eventId: eventId,
    eventTitle: eventData.title,
    createdAt: new Date(),
    eventDate: eventData.dateTime || new Date(),
    reminderSetting: reminderSetting,
    reminderSent: false
  });

  // Increment attendee count
  const eventRef = this.db.collection('events').doc(eventId);
  batch.update(eventRef, {
    attendees: firebase.firestore.FieldValue.increment(1)
  });

  // Commit the batch
  await batch.commit();
```

```
    return true;
  } catch (error) {
    console.error('Error RSVPing to event:', error);
    throw error;
  }
}

async cancelRsvp(eventId) {
  const user = firebase.auth().currentUser;
  if (!user) {
    throw new Error('User must be logged in to cancel RSVP');
  }

  try {
    // Find the attendee record
    const attendeeSnapshot = await this.db.collection('attendees')
      .where('userId', '==', user.uid)
      .where('eventId', '==', eventId)
      .get();

    if (attendeeSnapshot.empty) {
      throw new Error('You have not RSVP\'d to this event');
    }

    // Begin a batch write
    const batch = this.db.batch();

    // Delete attendee records
    attendeeSnapshot.forEach(doc => {
      batch.delete(doc.ref);
    });

    // Decrement attendee count
    const eventRef = this.db.collection('events').doc(eventId);
    batch.update(eventRef, {
      attendees: firebase.firestore.FieldValue.increment(-1)
    });

    // Commit the batch
    await batch.commit();

    return true;
  } catch (error) {
```

```
      console.error('Error canceling RSVP:', error);
      throw error;
    }
  }

  async getEventAttendees(eventId, limit = 20) {
    try {
      const snapshot = await this.db.collection('attendees')
        .where('eventId', '==', eventId)
        .limit(limit)
        .get();

      const userIds = snapshot.docs.map(doc => doc.data().userId);

      if (userIds.length === 0) return [];

      // Get user profiles
      const userDocs = await Promise.all(
        userIds.map(userId => this.db.collection('users').doc(userId).get())
      );

      return userDocs
        .filter(doc => doc.exists)
        .map(doc => ({
          id: doc.id,
          ...doc.data()
        }));
    } catch (error) {
      console.error('Error getting event attendees:', error);
      return [];
    }
  }
}

// Create a singleton instance
const eventService = new EventService();

export default eventService;
```

2.4. Implement RSVP and Attendance Tracking (Duration: 1 day)

```
javascript
```

```
// Create file: scripts/services/attendance-service.js
class AttendanceService {
  constructor() {
    this.db = firebase.firestore();
  }

  async getUserAttendingEvents(userId = null) {
    // Use current user if no userId provided
    const user = userId || firebase.auth().currentUser;
    if (!user) return [];

    try {
      const now = new Date();

      // Get events the user is attending that haven't happened yet
      const snapshot = await this.db.collection('attendees')
        .where('userId', '==', typeof user === 'string' ? user : user.uid)
        .where('eventDate', '>=', now)
        .orderBy('eventDate', 'asc')
        .get();

      const eventIds = snapshot.docs.map(doc => doc.data().eventId);

      if (eventIds.length === 0) return [];

      // Get the actual events
      const eventsSnapshot = await this.db.collection('events')
        .where(firebase.firestore.FieldPath.documentId(), 'in', eventIds)
        .get();

      return eventsSnapshot.docs.map(doc => ({
        id: doc.id,
        ...doc.data()
      }));
    } catch (error) {
      console.error('Error getting attending events:', error);
      return [];
    }
  }

  async getUserPastEvents(userId = null) {
    // Use current user if no userId provided
    const user = userId || firebase.auth().currentUser;
```

```
  if (!user) return [];

  try {
    const now = new Date();

    // Get events the user attended that have already happened
    const snapshot = await this.db.collection('attendees')
      .where('userId', '==', typeof user === 'string' ? user : user.uid)
      .where('eventDate', '<', now)
      .orderBy('eventDate', 'desc')
      .get();

    const eventIds = snapshot.docs.map(doc => doc.data().eventId);

    if (eventIds.length === 0) return [];

    // Get the actual events
    const eventsSnapshot = await this.db.collection('events')
      .where(firebase.firestore.FieldPath.documentId(), 'in', eventIds)
      .get();

    return eventsSnapshot.docs.map(doc => ({
      id: doc.id,
      ...doc.data()
    }));
  } catch (error) {
    console.error('Error getting past events:', error);
    return [];
  }
}

async checkInToEvent(eventId, checkInCode) {
  const user = firebase.auth().currentUser;
  if (!user) {
    throw new Error('User must be logged in to check in');
  }

  try {
    // Verify check-in code
    const eventDoc = await this.db.collection('events').doc(eventId).get();

    if (!eventDoc.exists) {
      throw new Error('Event not found');
    }
```

```
    const event = eventDoc.data();

    // Check if check-in code is valid
    if (!event.checkInCode || event.checkInCode !== checkInCode) {
      throw new Error('Invalid check-in code');
    }

    // Check if user is RSVP'd
    const attendeeSnapshot = await this.db.collection('attendees')
      .where('userId', '==', user.uid)
      .where('eventId', '==', eventId)
      .get();

    if (attendeeSnapshot.empty) {
      throw new Error('You have not RSVP\'d to this event');
    }

    // Update the attendee record to mark as checked in
    const batch = this.db.batch();

    attendeeSnapshot.forEach(doc => {
      batch.update(doc.ref, {
        checkedIn: true,
        checkInTime: new Date()
      });
    });

    // Commit the batch
    await batch.commit();

    return true;
  } catch (error) {
    console.error('Error checking in to event:', error);
    throw error;
  }
}

async generateCheckInCode(eventId) {
  const user = firebase.auth().currentUser;
  if (!user) {
    throw new Error('User must be logged in to generate check-in code');
  }
```

```
  try {
    // Check if user is the creator of the event
    const eventDoc = await this.db.collection('events').doc(eventId).get();

    if (!eventDoc.exists) {
      throw new Error('Event not found');
    }

    const event = eventDoc.data();

    if (event.createdBy !== user.uid) {
      throw new Error('Only the event organizer can generate check-in codes');
    }

    // Generate a 6-digit code
    const code = Math.floor(100000 + Math.random() * 900000).toString();

    // Update the event with the new code
    await this.db.collection('events').doc(eventId).update({
      checkInCode: code,
      checkInCodeExpires: new Date(Date.now() + 3600000) // 1 hour expiration
    });

    return code;
  } catch (error) {
    console.error('Error generating check-in code:', error);
    throw error;
  }
}

async getEventAttendees(eventId) {
  const user = firebase.auth().currentUser;
  if (!user) {
    throw new Error('User must be logged in to view attendees');
  }

  try {
    // Check if user is the creator or RSVP'd
    const eventDoc = await this.db.collection('events').doc(eventId).get();

    if (!eventDoc.exists) {
      throw new Error('Event not found');
    }
```

```
const event = eventDoc.data();

// Check if user is the organizer or an attendee
const isOrganizer = event.createdBy === user.uid;

if (!isOrganizer) {
  const attendeeSnapshot = await this.db.collection('attendees')
    .where('userId', '==', user.uid)
    .where('eventId', '==', eventId)
    .get();

  if (attendeeSnapshot.empty) {
    throw new Error('You do not have permission to view the attendees list');
  }
}

// Get all attendees
const attendeesSnapshot = await this.db.collection('attendees')
  .where('eventId', '==', eventId)
  .get();

const attendeeUserIds = attendeesSnapshot.docs.map(doc => doc.data().userId);

if (attendeeUserIds.length === 0) return [];

// Get attendee profiles
const attendeeProfiles = [];

// Batch attendee IDs in groups of 10 (Firestore limitation)
for (let i = 0; i < attendeeUserIds.length; i += 10) {
  const batch = attendeeUserIds.slice(i, i + 10);

  const usersSnapshot = await this.db.collection('users')
    .where(firebase.firestore.FieldPath.documentId(), 'in', batch)
    .get();

  usersSnapshot.forEach(doc => {
    attendeeProfiles.push({
      id: doc.id,
      ...doc.data()
    });
  });
}
```

```
      return attendeeProfiles;
    } catch (error) {
      console.error('Error getting event attendees:', error);
      throw error;
    }
  }
}

// Create a singleton instance
const attendanceService = new AttendanceService();

export default attendanceService;
```

3. Map and Location Features (Duration: 6 days)

3.1. Implement Event Map Page (Duration: 3 days)

```
html
```

```
<!-- Create file: pages/app/map.html -->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Event Map - The Play</title>
  <!-- Include your standard CSS and JS files here -->
  <link href="https://cdnjs.cloudflare.com/ajax/libs/tailwindcss/2.2.19/tailwind.min.css" rel="
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0-beta3/
  <link rel="stylesheet" href="../../assets/styles/main.css">
  <link rel="stylesheet" href="../../assets/styles/variables.css">
  <link rel="stylesheet" href="../../assets/styles/layout.css">
  <link rel="stylesheet" href="../../assets/styles/components.css">

  <!-- Firebase Libraries -->
  <script src="https://www.gstatic.com/firebasejs/9.22.2/firebase-app-compat.js"></script>
  <script src="https://www.gstatic.com/firebasejs/9.22.2/firebase-auth-compat.js"></script>
  <script src="https://www.gstatic.com/firebasejs/9.22.2/firebase-firestore-compat.js"></script
  <script src="../../scripts/firebase/config.js"></script>

  <!-- Leaflet Map CSS -->
  <link rel="stylesheet" href="https://unpkg.com/leaflet@1.9.4/dist/leaflet.css" />

  <style>
    .map-container {
      height: calc(100vh - 6rem);
    }

    .event-popup .leaflet-popup-content-wrapper {
      border-radius: 12px;
      box-shadow: 0 4px 15px rgba(0, 0, 0, 0.1);
      padding: 0;
      overflow: hidden;
    }

    .event-popup .leaflet-popup-content {
      margin: 0;
      width: 250px !important;
    }

    .event-popup .leaflet-popup-tip {
      box-shadow: 0 4px 15px rgba(0, 0, 0, 0.1);
```

```
}

.map-filter-panel {
  z-index: 1000;
  top: 1rem;
  left: 1rem;
  max-width: 300px;
}

.map-sidebar {
  height: calc(100vh - 6rem);
  overflow-y: auto;
}

.custom-marker {
  display: flex;
  align-items: center;
  justify-content: center;
  width: 36px;
  height: 36px;
  border-radius: 50%;
  background-color: white;
  box-shadow: 0 2px 5px rgba(0, 0, 0, 0.2);
  border: 2px solid;
  font-size: 16px;
}

.marker-music { border-color: #6C63FF; color: #6C63FF; }
.marker-art { border-color: #3B82F6; color: #3B82F6
```