</div> `;

```
    // Add click event
    element.addEventListener('click', () => {
      window.location.href = `event-detail.html?id=${event.id}`;
    });

    return element;
  }

  function showLoadingState() {
    document.getElementById('initial-state').classList.add('hidden');
    document.getElementById('no-results').classList.add('hidden');
    document.getElementById('results-grid').classList.add('hidden');
    document.getElementById('loading-state').classList.remove('hidden');
  }

  function showNoResults() {
    document.getElementById('initial-state').classList.add('hidden');
    document.getElementById('loading-state').classList.add('hidden');
    document.getElementById('results-grid').classList.add('hidden');
    document.getElementById('no-results').classList.remove('hidden');
  }

  function updateSearchURL(query, category, date, price) {
    // Create a URL object
    const url = new URL(window.location.href);

    // Clear existing parameters
    url.search = '';

    // Add parameters if they have values
    if (query) url.searchParams.set('q', query);
    if (category) url.searchParams.set('category', category);
    if (date) url.searchParams.set('date', date);
    if (price) url.searchParams.set('price', price);

    // Update the URL without reloading the page
    window.history.pushState({}, '', url);
  }
```

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

4.2. Implement Search Bar in Header (Duration: 1 day)

html

```html
<!-- Modify: components/header.html -->
<!-- Add this inside the header, before the Create Event button: -->
<div class="hidden md:block relative mr-4">
  <form id="search-form" action="../../pages/app/search.html" method="get">
    <input type="text" name="q" placeholder="Search events..." class="w-64 py-2 px-10 rounded-f
    <i class="fas fa-search absolute left-3 top-3 text-gray-400"></i>
    <button type="submit" class="absolute right-3 top-2 text-gray-500 hover:text-purple-600">
      <i class="fas fa-arrow-right"></i>
    </button>
  </form>
</div>
```

4.3. Implement Event Recommendations (Duration: 2 days)

```
javascript
```

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

  async getUserPreferences() {
    const user = firebase.auth().currentUser;
    if (!user) return null;

    try {
      const userDoc = await this.db.collection('users').doc(user.uid).get();

      if (!userDoc.exists) {
        return null;
      }

      const userData = userDoc.data();

      return userData.preferences || {};
    } catch (error) {
      console.error('Error getting user preferences:', error);
      return null;
    }
  }

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

    try {
      await this.db.collection('users').doc(user.uid).update({
        'preferences': preferences,
        updatedAt: new Date()
      });

      return true;
    } catch (error) {
      console.error('Error updating user preferences:', error);
      return false;
    }
  }
```

```javascript
async getRecommendedEvents(limit = 6) {
  const user = firebase.auth().currentUser;
  if (!user) return [];

  try {
    // Get user preferences
    const preferences = await this.getUserPreferences();

    // Get user's bookmarked events to build interests
    const bookmarksSnapshot = await this.db.collection('bookmarks')
      .where('userId', '==', user.uid)
      .get();

    const bookmarkedEventIds = bookmarksSnapshot.docs.map(doc => doc.data().eventId);

    // Get user's attended events
    const attendanceSnapshot = await this.db.collection('attendees')
      .where('userId', '==', user.uid)
      .get();

    const attendedEventIds = attendanceSnapshot.docs.map(doc => doc.data().eventId);

    // Combine all event IDs to analyze
    const allEventIds = [...new Set([...bookmarkedEventIds, ...attendedEventIds])];

    if (allEventIds.length === 0) {
      // User has no history, return trending events
      return this.getTrendingEvents(limit);
    }

    // Get event details in batches (Firestore limitation)
    const eventDetails = [];

    for (let i = 0; i < allEventIds.length; i += 10) {
      const batch = allEventIds.slice(i, i + 10);

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

      eventsSnapshot.forEach(doc => {
        eventDetails.push({
          id: doc.id,
          ...doc.data()
        });
      });
    }

    // Build category preferences
    const categoryCount = {};

    eventDetails.forEach(event => {
      if (event.category) {
        categoryCount[event.category] = (categoryCount[event.category] || 0) + 1;
      }
    });

    // Sort categories by count
    const sortedCategories = Object.entries(categoryCount)
      .sort((a, b) => b[1] - a[1])
      .map(entry => entry[0]);

    // If user has explicit preferences, prioritize those
    let preferredCategories = sortedCategories;

    if (preferences && preferences.eventTypes && preferences.eventTypes.length > 0) {
      // Combine explicit preferences with inferred ones, prioritizing explicit
      preferredCategories = [
        ...preferences.eventTypes,
        ...sortedCategories.filter(cat => !preferences.eventTypes.includes(cat))
      ];
    }

    // Get upcoming events in preferred categories
    const now = new Date();

    let recommendedEvents = [];

    // Try each preferred category
    for (const category of preferredCategories) {
      const categoryEventsSnapshot = await this.db.collection('events')
        .where('category', '==', category)
        .where('dateTime', '>=', now)
        .limit(limit - recommendedEvents.length)
        .get();

      categoryEventsSnapshot.forEach(doc => {
        // Skip events user has already bookmarked or attended
```

```
          if (!allEventIds.includes(doc.id)) {
            recommendedEvents.push({
              id: doc.id,
              ...doc.data()
            });
          }
        });

        // If we have enough events, stop querying
        if (recommendedEvents.length >= limit) {
          break;
        }
      }

      // If we still need more events, get trending events
      if (recommendedEvents.length < limit) {
        const trendingEvents = await this.getTrendingEvents(limit - recommendedEvents.length);

        // Add trending events, avoiding duplicates
        trendingEvents.forEach(event => {
          if (!recommendedEvents.some(recEvent => recEvent.id === event.id)) {
            recommendedEvents.push(event);
          }
        });
      }

      return recommendedEvents.slice(0, limit);
    } catch (error) {
      console.error('Error getting recommended events:', error);
      return this.getTrendingEvents(limit);
    }
  }

  async getTrendingEvents(limit = 6) {
    try {
      // Get events with high attendance
      const eventsSnapshot = await this.db.collection('events')
        .where('dateTime', '>=', new Date())
        .orderBy('dateTime', 'asc')
        .orderBy('attendees', 'desc')
        .limit(limit)
        .get();

      const events = [];
      eventsSnapshot.forEach(doc => {
        events.push({
          id: doc.id,
          ...doc.data()
        });
      });

      return events;
    } catch (error) {
      console.error('Error getting trending events:', error);
      return [];
    }
  }

  async getEventsForYou(limit = 10) {
    try {
      // Get recommended events
      const recommendedEvents = await this.getRecommendedEvents(limit / 2);

      // Get nearby events (we'll implement this in a future update)
      const nearbyEvents = [];

      // Get trending events to fill in if needed
      const numMoreNeeded = limit - recommendedEvents.length - nearbyEvents.length;

      let trendingEvents = [];

      if (numMoreNeeded > 0) {
        trendingEvents = await this.getTrendingEvents(numMoreNeeded);
      }

      // Combine all events, removing duplicates
      const allEvents = [...recommendedEvents];

      // Add nearby events, avoiding duplicates
      nearbyEvents.forEach(event => {
        if (!allEvents.some(e => e.id === event.id)) {
          allEvents.push(event);
        }
      });

      // Add trending events, avoiding duplicates
      trendingEvents.forEach(event => {
        if (!allEvents.some(e => e.id === event.id)) {
          allEvents.push(event);
        }
      });

      return allEvents.slice(0, limit);
    } catch (error) {
      console.error('Error getting events for you:', error);
      return [];
    }
  }

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

    try {
      // Save interaction to Firestore
      await this.db.collection('userInteractions').add({
        userId: user.uid,
        eventId: eventId,
        type: interactionType,
        timestamp: new Date()
      });

      return true;
    } catch (error) {
      console.error('Error saving event interaction:', error);
      return false;
    }
  }
}

// Create a singleton instance
const recommendationService = new RecommendationService();

export default recommendationService;
```

```
javascript
```

```
// Create file: scripts/components/recommended-events.js
import recommendationService from '../services/recommendation-service.js';
import { formatEventDate } from '../utils/date-format.js';
import errorHandler from '../utils/error-handler.js';

class RecommendedEventsComponent {
  constructor(containerId, options = {}) {
    this.containerId = containerId;
    this.container = document.getElementById(containerId);
    this.options = {
      limit: 6,
      title: 'Recommended For You',
      showViewAll: true,
      viewAllURL: 'index.html',
      ...options
    };

    if (!this.container) {
      console.error(`Container with ID ${containerId} not found`);
      return;
    }

    this.initialize();
  }

  async initialize() {
    try {
      // Show loading state
      this.showLoading();

      // Load recommended events
      const events = await recommendationService.getRecommendedEvents(this.options.limit);

      // Update UI
      this.updateUI(events);
    } catch (error) {
      console.error('Error initializing recommended events component:', error);
      this.showError('Failed to load recommended events');
    }
  }

  updateUI(events) {
    // Clear container
    this.container.innerHTML = '';

    // Check if we have events
    if (events.length === 0) {
      this.showNoEvents();
      return;
    }

    // Add heading
    const heading = document.createElement('h2');
    heading.className = 'text-xl font-bold mb-4';
    heading.textContent = this.options.title;
    this.container.appendChild(heading);

    // Create events grid
    const grid = document.createElement('div');
    grid.className = 'grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-4';

    // Add events to grid
    events.forEach(event => {
      const eventElement = this.createEventElement(event);
      grid.appendChild(eventElement);
    });

    // Add grid to container
    this.container.appendChild(grid);

    // Add "View All" button if requested
    if (this.options.showViewAll) {
      const viewAllBtn = document.createElement('div');
      viewAllBtn.className = 'text-center mt-6';
      viewAllBtn.innerHTML = `
        <a href="${this.options.viewAllURL}" class="btn-primary inline-block">
          View All
        </a>
      `;
      this.container.appendChild(viewAllBtn);
    }
  }

  createEventElement(event) {
    const element = document.createElement('div');
    element.className = 'bg-white rounded-lg shadow-sm overflow-hidden';
    // Format date
    let dateStr = 'Date TBD';
    if (event.dateTime) {
      const dateTime = new Date(event.dateTime.seconds * 1000);
      dateStr = formatEventDate(dateTime);
    }

    element.innerHTML = `
      <div class="h-36 bg-gray-200 relative">
        <img src="${event.imageUrl || '/api/placeholder/400/250'}" alt="${event.title}" class="
        <div class="absolute top-2 right-2 vibe-score text-xs">
          <i class="fas fa-fire mr-1"></i> ${event.score || '8.5'}
        </div>
      </div>
      <div class="p-3">
        <h3 class="font-bold text-sm mb-1">${event.title}</h3>
        <p class="text-xs text-gray-600 mb-1">
          <i class="fas fa-calendar-alt mr-1"></i> ${dateStr}
        </p>
        <p class="text-xs text-gray-600 truncate">
          <i class="fas fa-map-marker-alt mr-1"></i> ${event.location || 'Location TBD'}
        </p>
      </div>
    `;

    // Add click event
    element.addEventListener('click', () => {
      // Record the click for better recommendations
      recommendationService.saveEventInteraction(event.id, 'click');

      // Navigate to event detail page
      window.location.href = `event-detail.html?id=${event.id}`;
    });

    return element;
  }

  showLoading() {
    this.container.innerHTML = `
      <div class="text-center py-8">
        <div class="animate-spin rounded-full h-8 w-8 border-t-2 border-b-2 border-primary mx-a
        <p class="text-gray-600">Finding events for you...</p>
      </div>
    `;
  }

  showNoEvents() {
    this.container.innerHTML = `
      <div class="text-center py-8">
        <p class="text-gray-600 mb-4">No recommended events found.</p>
        <a href="map.html" class="btn-primary inline-block">
          Explore All Events
        </a>
      </div>
    `;
  }

  showError(message) {
    this.container.innerHTML = `
      <div class="text-center py-8">
        <p class="text-gray-600 mb-4">${message}</p>
        <button class="btn-primary inline-block" onclick="location.reload()">
          Try Again
        </button>
      </div>
    `;
  }
}

export default RecommendedEventsComponent;
```

5. Push Notifications (Duration: 4 days)

5.1. Set Up Firebase Cloud Messaging (Duration: 1 day)

```
javascript
```

```
// Create file: scripts/services/notification-service.js
class NotificationService {
  constructor() {
    this.db = firebase.firestore();
    this.messaging = null;
    this.initialized = false;
    this.permission = 'default';
  }

  async initialize() {
    if (this.initialized) return this.permission;

    try {
      // Check if Firebase Messaging is available
      if (!firebase.messaging) {
        console.warn('Firebase Messaging is not available');
        return 'not-supported';
      }

      this.messaging = firebase.messaging();

      // Get permission status
      this.permission = Notification.permission;

      // Set up service worker if permission granted
      if (this.permission === 'granted') {
        await this.setupServiceWorker();
      }

      this.initialized = true;
      return this.permission;
    } catch (error) {
      console.error('Error initializing notifications:', error);
      return 'error';
    }
  }

  async requestPermission() {
    try {
      // Initialize if not already
      if (!this.initialized) {
        await this.initialize();
      }
      // Check if messaging is available
      if (!this.messaging) {
        return 'not-supported';
      }

      // Request permission
      const permission = await Notification.requestPermission();
      this.permission = permission;

      // If permission granted, set up service worker
      if (permission === 'granted') {
        await this.setupServiceWorker();
      }

      return permission;
    } catch (error) {
      console.error('Error requesting permission:', error);
      return 'error';
    }
  }

  async setupServiceWorker() {
    try {
      // Register service worker
      const registration = await navigator.serviceWorker.register('/firebase-messaging-sw.js');

      // Set up messaging
      this.messaging.useServiceWorker(registration);

      // Get token
      const token = await this.messaging.getToken();

      // Save token to Firestore
      await this.saveToken(token);

      // Handle token refresh
      this.messaging.onTokenRefresh(async () => {
        const refreshedToken = await this.messaging.getToken();
        await this.saveToken(refreshedToken);
      });

      // Handle foreground messages
      this.messaging.onMessage((payload) => {
        console.log('Message received in foreground:', payload);

        // Show notification
        const notification = new Notification(payload.notification.title, {
          body: payload.notification.body,
          icon: payload.notification.icon || '/icon.png'
        });

        // Handle notification click
        notification.onclick = () => {
          if (payload.data && payload.data.url) {
            window.open(payload.data.url, '_blank');
          }
          notification.close();
        };
      });

      return 'success';
    } catch (error) {
      console.error('Error setting up service worker:', error);
      return 'error';
    }
  }

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

    try {
      // Save token to Firestore
      await this.db.collection('userNotifications').doc(user.uid).set({
        token: token,
        updatedAt: new Date(),
        enabled: true,
        platform: 'web'
      }, { merge: true });

      return true;
    } catch (error) {
      console.error('Error saving token:', error);
      return false;
    }
  }
```

```
async updateNotificationPreferences(preferences) {
  const user = firebase.auth().currentUser;
  if (!user) return false;

  try {
    // Update preferences in Firestore
    await this.db.collection('userNotifications').doc(user.uid).update({
      preferences: preferences,
      updatedAt: new Date()
    });

    return true;
  } catch (error) {
    console.error('Error updating notification preferences:', error);
    return false;
  }
}

async getNotificationPreferences() {
  const user = firebase.auth().currentUser;
  if (!user) return null;

  try {
    // Get preferences from Firestore
    const doc = await this.db.collection('userNotifications').doc(user.uid).get();

    if (!doc.exists) {
      return {
        enabled: true,
        eventReminders: true,
        vibeUpdates: true,
        newEvents: true
      };
    }

    const data = doc.data();

    // Return preferences
    return {
      enabled: data.enabled ?? true,
      eventReminders: data.preferences?.eventReminders ?? true,
      vibeUpdates: data.preferences?.vibeUpdates ?? true,
      newEvents: data.preferences?.newEvents ?? true
    };
    } catch (error) {
      console.error('Error getting notification preferences:', error);
      return null;
    }
  }

  async sendEventReminder(eventId, userId, reminderType) {
    // This would typically be done on the server, but we'll define the interface here
    // In a real app, you would create a Cloud Function to handle this

    try {
      // Create a reminder document
      await this.db.collection('notifications').add({
        eventId: eventId,
        userId: userId,
        type: 'event-reminder',
        reminderType: reminderType, // 'day' or 'hour'
        status: 'pending',
        createdAt: new Date()
      });

      return true;
    } catch (error) {
      console.error('Error scheduling event reminder:', error);
      return false;
    }
  }
}

// Create a singleton instance
const notificationService = new NotificationService();

export default notificationService;
```

5.2. Create Notification Preferences UI (Duration: 1 day)

html

```html
<!-- Create file: components/modals/notification-settings-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">Notification Settings</h3>
      <button class="text-gray-500 hover:text-gray-700 close-modal-btn">
        <i class="fas fa-times"></i>
      </button>
    </div>

    <div id="notifications-not-supported" class="hidden">
      <p class="text-gray-700 mb-4">Push notifications are not supported in your browser.</p>
      <p class="text-gray-600 text-sm mb-4">Please use a modern browser like Chrome, Firefox, c
    </div>

    <div id="notifications-permission" class="hidden">
      <p class="text-gray-700 mb-4">The Play would like to send you notifications about your ev

      <div class="bg-gray-100 p-4 rounded-lg mb-6">
        <div class="flex items-center mb-2">
          <i class="fas fa-bell text-purple-600 mr-3"></i>
          <p class="text-gray-800">Get notified about:</p>
        </div>
        <ul class="pl-9 text-gray-700 text-sm space-y-2">
          <li>• Event reminders</li>
          <li>• Vibe check updates</li>
          <li>• New events that match your interests</li>
        </ul>
      </div>

      <div class="flex justify-end">
        <button id="enable-notifications" class="btn-primary px-6 py-3">Enable Notifications</t
      </div>
    </div>

    <div id="notifications-settings" class="hidden">
      <p class="text-gray-700 mb-4">Manage your notification preferences below.</p>

      <div class="space-y-4 mb-6">
        <div class="flex items-center justify-between">
          <div>
            <p class="font-medium">Enable All Notifications</p>
            <p class="text-sm text-gray-600">Turn all notifications on or off</p>
          </div>
          <label class="switch">
            <input type="checkbox" id="notifications-enabled">
            <span class="slider round"></span>
          </label>
        </div>

        <div class="border-t border-gray-200 pt-4">
          <p class="font-medium mb-3">Notification Types</p>

          <div class="space-y-3">
            <div class="flex items-center justify-between">
              <p class="text-gray-700">Event Reminders</p>
              <label class="switch">
                <input type="checkbox" id="event-reminders">
                <span class="slider round"></span>
              </label>
            </div>

            <div class="flex items-center justify-between">
              <p class="text-gray-700">Vibe Check Updates</p>
              <label class="switch">
                <input type="checkbox" id="vibe-updates">
                <span class="slider round"></span>
              </label>
            </div>

            <div class="flex items-center justify-between">
              <p class="text-gray-700">New Event Recommendations</p>
              <label class="switch">
                <input type="checkbox" id="new-events">
                <span class="slider round"></span>
              </label>
            </div>
          </div>
        </div>
      </div>

      <div class="flex justify-end">
        <button id="save-notification-settings" class="btn-primary px-6 py-3">Save Settings</bu
      </div>
    </div>

    <div id="notifications-blocked" class="hidden">
```

```
      <p class="text-gray-700 mb-4">Notifications are currently blocked for this site.</p>
      <p class="text-gray-600 text-sm mb-6">To enable notifications, please update your browser

      <div class="bg-gray-100 p-4 rounded-lg mb-6">
        <h4 class="font-medium mb-2">How to enable notifications:</h4>
        <ol class="pl-5 text-gray-700 text-sm space-y-2">
          <li>1. Click the lock/info icon in your browser's address bar</li>
          <li>2. Find "Notifications" in the site settings</li>
          <li>3. Change the setting from "Block" to "Allow"</li>
          <li>4. Refresh this page</li>
        </ol>
      </div>

      <div class="flex justify-end">
        <button id="check-permission-again" class="btn-primary px-6 py-3">Check Again</button>
      </div>
    </div>
  </div>
</div>

<style>
  /* Toggle Switch Styles */
  .switch {
    position: relative;
    display: inline-block;
    width: 48px;
    height: 24px;
  }

  .switch input {
    opacity: 0;
    width: 0;
    height: 0;
  }

  .slider {
    position: absolute;
    cursor: pointer;
    top: 0;
    left: 0;
    right: 0;
    bottom: 0;
    background-color: #ccc;
    transition: .4s;
  }

  .slider:before {
    position: absolute;
    content: "";
    height: 18px;
    width: 18px;
    left: 3px;
    bottom: 3px;
    background-color: white;
    transition: .4s;
  }

  input:checked + .slider {
    background-color: var(--primary);
  }

  input:focus + .slider {
    box-shadow: 0 0 1px var(--primary);
  }

  input:checked + .slider:before {
    transform: translateX(24px);
  }

  .slider.round {
    border-radius: 24px;
  }

  .slider.round:before {
    border-radius: 50%;
  }
</style>

<script type="module">
  import notificationService from '../../scripts/services/notification-service.js';
  import errorHandler from '../../scripts/utils/error-handler.js';

  document.addEventListener('DOMContentLoaded', async function() {
    // Initialize notification service
    const permissionStatus = await notificationService.initialize();

    // Set up UI based on permission status
    updateUI(permissionStatus);
    // Close modal functionality
    document.querySelectorAll('#notificationSettingsModal .close-modal-btn').forEach(btn => {
      btn.addEventListener('click', function() {
        closeModal();
      });
    });

    // Close when clicking outside
    document.getElementById('notificationSettingsModal').addEventListener('click', function(e)
      if (e.target === this) {
        closeModal();
      }
    });

    // Enable notifications button
    document.getElementById('enable-notifications').addEventListener('click', async function()
      const result = await notificationService.requestPermission();
      updateUI(result);

      if (result === 'granted') {
        errorHandler.showToast('Notifications enabled successfully!', 'success');
      } else if (result === 'denied') {
        errorHandler.showToast('Notification permission denied', 'error');
      }
    });

    // Check permission again button
    document.getElementById('check-permission-again').addEventListener('click', async function(
      const result = await notificationService.initialize();
      updateUI(result);
    });

    // Save settings button
    document.getElementById('save-notification-settings').addEventListener('click', async funct
      // Show loading state
      this.disabled = true;
      this.innerHTML = '<i class="fas fa-spinner fa-spin mr-2"></i> Saving...';

      // Get settings
      const enabled = document.getElementById('notifications-enabled').checked;
      const eventReminders = document.getElementById('event-reminders').checked;
      const vibeUpdates = document.getElementById('vibe-updates').checked;
      const newEvents = document.getElementById('new-events').checked;
```

```
// Update preferences
const success = await notificationService.updateNotificationPreferences({
  eventReminders,
  vibeUpdates,
  newEvents
});

// Reset button
this.disabled = false;
```