this.innerHTML = 'Save Settings';

```
    if (success) {
      // Update enabled status
      await notificationService.saveToken(null, enabled);

      // Show success message
      errorHandler.showToast('Notification settings saved', 'success');

      // Close modal
      closeModal();
    } else {
      errorHandler.showToast('Failed to save notification settings', 'error');
    }
  });

  // Master toggle for all notifications
  document.getElementById('notifications-enabled').addEventListener('change', function() {
    const enabled = this.checked;

    // Update other toggles
    document.getElementById('event-reminders').disabled = !enabled;
    document.getElementById('vibe-updates').disabled = !enabled;
    document.getElementById('new-events').disabled = !enabled;
  });

  // Load notification preferences
  await loadNotificationPreferences();

});
```

async function updateUI(permissionStatus) {
// Hide all sections
document.getElementById('notifications-not-supported').classList.add('hidden');
document.getElementById('notifications-permission').classList.add('hidden');
document.getElementById('notifications-settings').classList.add('hidden');
document.getElementById('notifications-blocked').classList.add('hidden');

```
  // Show appropriate section based on permission status
  if (permissionStatus === 'not-supported') {
    document.getElementById('notifications-not-supported').classList.remove('hidden');
  } else if (permissionStatus === 'default') {
    document.getElementById('notifications-permission').classList.remove('hidden');
  } else if (permissionStatus === 'granted') {
    document.getElementById('notifications-settings').classList.remove('hidden');
    await loadNotificationPreferences();
  } else if (permissionStatus === 'denied') {
    document.getElementById('notifications-blocked').classList.remove('hidden');
  }


}

async function loadNotificationPreferences() {
try {
const preferences = await notificationService.getNotificationPreferences();


    if (preferences) {
      // Update toggles
      document.getElementById('notifications-enabled').checked = preferences.enabled;
      document.getElementById('event-reminders').checked = preferences.eventReminders;
      document.getElementById('vibe-updates').checked = preferences.vibeUpdates;
      document.getElementById('new-events').checked = preferences.newEvents;

      // Update disabled state
      document.getElementById('event-reminders').disabled = !preferences.enabled;
      document.getElementById('vibe-updates').disabled = !preferences.enabled;
      document.getElementById('new-events').disabled = !preferences.enabled;
    }
  } catch (error) {
    console.error('Error loading notification preferences:', error);
  }


}

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

```
}
</script>
```

5.3. Create Service Worker for Notifications (Duration: 1 day)

```javascript
// Create file: public/firebase-messaging-sw.js
importScripts('https://www.gstatic.com/firebasejs/9.22.2/firebase-app-compat.js');
importScripts('https://www.gstatic.com/firebasejs/9.22.2/firebase-messaging-compat.js');

// Initialize Firebase
firebase.initializeApp({
  apiKey: "AIzaSyBOPjdRIDVst4uZg6oqNPLhlqgj0XwqPH8",
  authDomain: "the-play-8f454.firebaseapp.com",
  projectId: "the-play-8f454",
  storageBucket: "the-play-8f454.firebasestorage.app",
  messagingSenderId: "919736772232",
  appId: "1:919736772232:web:5d52b627785e5569a7e13d",
  measurementId: "G-KYB0MJ4VKS"
});

// Get messaging instance
const messaging = firebase.messaging();

// Handle background messages
messaging.onBackgroundMessage((payload) => {
  console.log('Received background message:', payload);

  // Customize notification here
  const notification = payload.notification;

  const notificationTitle = notification.title;
  const notificationOptions = {
    body: notification.body,
    icon: notification.icon || '/icon.png'
  };

  self.registration.showNotification(notificationTitle, notificationOptions);
});

// Handle notification click
self.addEventListener('notificationclick', (event) => {
  console.log('Notification click:', event);

  event.notification.close();
```

```
  // This looks to see if the current is already open and focuses if it is
  event.waitUntil(
    clients.matchAll({
      type: 'window'
    })
    .then((clientList) => {
      // Check if there is already a window/tab open with the target URL
      const url = event.notification.data?.url || '/';
      const hadWindowToFocus = clientList.some((client) => {
        if (client.url === url && 'focus' in client) {
          client.focus();
          return true;
        }
        return false;
      });

      // If no existing window, open a new one
      if (!hadWindowToFocus) {
        clients.openWindow(url).catch(err => console.error('Failed to open window:', err));
      }
    })
  );
});
```

5.4. Set Up Event Reminders (Duration: 1 day)

```
javascript
```

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

  async scheduleEventReminder(eventId, reminderType = 'day') {
    const user = firebase.auth().currentUser;
    if (!user) return false;

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

      if (!eventDoc.exists) {
        console.error('Event not found');
        return false;
      }

      const eventData = eventDoc.data();
      const eventDate = eventData.dateTime ? new Date(eventData.dateTime.seconds * 1000) : null

      if (!eventDate) {
        console.error('Event has no date');
        return false;
      }

      // Calculate reminder time
      let reminderTime;

      if (reminderType === 'day') {
        // 24 hours before the event
        reminderTime = new Date(eventDate.getTime() - 24 * 60 * 60 * 1000);
      } else if (reminderType === 'hour') {
        // 1 hour before the event
        reminderTime = new Date(eventDate.getTime() - 60 * 60 * 1000);
      } else {
        console.error('Invalid reminder type');
        return false;
      }

      // Create reminder document
      await this.db.collection('reminders').add({
```

```
      userId: user.uid,
      eventId: eventId,
      eventTitle: eventData.title,
      eventDate: eventData.dateTime,
      reminderType: reminderType,
      reminderTime: reminderTime,
      sent: false,
      createdAt: new Date()
    });

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

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

  try {
    // Find existing reminders
    const remindersSnapshot = await this.db.collection('reminders')
      .where('userId', '==', user.uid)
      .where('eventId', '==', eventId)
      .get();

    if (remindersSnapshot.empty) {
      return true; // No reminders to cancel
    }

    // Delete each reminder
    const batch = this.db.batch();

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

    await batch.commit();

    return true;
  } catch (error) {
    console.error('Error canceling reminder:', error);
```

```
      return false;
    }
  }

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

    try {
      // Get upcoming reminders
      const now = new Date();

      const remindersSnapshot = await this.db.collection('reminders')
        .where('userId', '==', user.uid)
        .where('reminderTime', '>', now)
        .orderBy('reminderTime', 'asc')
        .get();

      const reminders = [];

      remindersSnapshot.forEach(doc => {
        reminders.push({
          id: doc.id,
          ...doc.data()
        });
      });

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

// Create a singleton instance
const reminderService = new ReminderService();

export default reminderService;
```

Testing and Documentation (Duration: 3 days)

1. Create Test Users and Data

- Set up test user accounts with different profiles
- Create a diverse set of test events across categories
- Generate test vibe checks for events

2. Test Cross-Browser Compatibility

- Test on Chrome, Firefox, Safari, and Edge
- Verify responsive design on different screen sizes
- Test on iOS and Android mobile browsers

3. Document API and Components

```
javascript
```

```
// Create file: docs/api-reference.md
# The Play API Reference

## Authentication Module

### `authentication.signUp(email, password)`
Creates a new user account.

**Parameters:**
- `email` (string): User's email address
- `password` (string): User's password

**Returns:**
- Promise resolving to an object with `success` (boolean) and `user` (if successful) or `error`

### `authentication.logIn(email, password)`
Logs in an existing user.

**Parameters:**
- `email` (string): User's email address
- `password` (string): User's password

**Returns:**
- Promise resolving to an object with `success` (boolean) and `user` (if successful) or `error`

### `authentication.signInWithGoogle()`
Initiates Google sign-in.

**Returns:**
- Promise resolving to an object with `success` (boolean) and `user` (if successful) or `error`

### `authentication.signInWithFacebook()`
Initiates Facebook sign-in.

**Returns:**
- Promise resolving to an object with `success` (boolean) and `user` (if successful) or `error`

### `authentication.logOut()`
Logs out the current user.

**Returns:**
- Promise resolving to an object with `success` (boolean) or `error` (if failed)
```

User Profile Module

`userProfile.initialize()`
Initializes the user profile and loads user data.

Returns:
- Promise resolving to the user data

`userProfile.getDisplayName()`
Gets the user's display name.

Returns:
- String with the user's display name or a default value

`userProfile.getPhotoURL()`
Gets the user's profile photo URL.

Returns:
- String with the user's photo URL or a default value

`userProfile.updateProfile(data)`
Updates the user's profile information.

Parameters:
- `data` (object): Object containing profile fields to update

Returns:
- Promise resolving to the updated user data

Event Service

`eventService.getEventById(eventId)`
Gets an event by its ID.

Parameters:
- `eventId` (string): ID of the event to retrieve

Returns:
- Promise resolving to the event data

`eventService.getEvents(filters, limit)`
Gets a list of events based on filters.

Parameters:

- `filters` (object, optional): Object containing filter criteria
- `limit` (number, optional): Maximum number of events to return

Returns:
- Promise resolving to an array of events

`eventService.createEvent(eventData, imageFile)`
Creates a new event.

Parameters:
- `eventData` (object): Event details
- `imageFile` (File, optional): Event image file

Returns:
- Promise resolving to the created event

Location Service

`locationService.getUserLocation()`
Gets the user's current location.

Returns:
- Promise resolving to location object with `latitude`, `longitude`, and `accuracy`

`locationService.geocodeAddress(address)`
Converts an address to coordinates.

Parameters:
- `address` (string): Address to geocode

Returns:
- Promise resolving to location object with `latitude` and `longitude`

`locationService.findNearbyEvents(radius)`
Finds events near the user's location.

Parameters:
- `radius` (number, optional): Search radius in kilometers

Returns:
- Promise resolving to an array of events

Notification Service

`notificationService.initialize()`
Initializes the notification service.

Returns:
- Promise resolving to the permission status

`notificationService.requestPermission()`
Requests notification permission from the user.

Returns:
- Promise resolving to the permission status

`notificationService.updateNotificationPreferences(preferences)`
Updates the user's notification preferences.

Parameters:
- `preferences` (object): Notification preference settings

Returns:
- Promise resolving to a boolean indicating success

Recommendation Service

`recommendationService.getRecommendedEvents(limit)`
Gets events recommended for the user.

Parameters:
- `limit` (number, optional): Maximum number of events to return

Returns:
- Promise resolving to an array of recommended events

`recommendationService.getTrendingEvents(limit)`
Gets trending events.

Parameters:
- `limit` (number, optional): Maximum number of events to return

Returns:
- Promise resolving to an array of trending events

```
javascript

// Create file: docs/components-reference.md
# The Play Components Reference

## Header Component
**File:** `components/header.html`

Main navigation header for the application, including the logo, navigation links, search bar, a

**Dependencies:**
- `scripts/auth/authentication.js`

**Usage:**
```javascript
import { loadComponent } from '../../scripts/utils/component-loader.js';

// Load the header component
loadComponent('header-container', '../../components/header.html');
```

Event Form Component

File: `components/events/event-form.html`

Form for creating and editing events, with fields for title, description, date, time, location, price, image upload, and category.

Dependencies:

- `scripts/components/event-creation.js`

Usage:
```

```javascript
import { loadComponent } from '../../scripts/utils/component-loader.js';
import eventCreator from '../../scripts/components/event-creation.js';

// Load the event form component
loadComponent('event-form-container', '../../components/events/event-form.html')
  .then(() => {
    // Initialize event creator
    eventCreator.initialize();
  });
```

Vibe Check Modal

File: `components/modals/vibe-check-modal.html`

Modal for posting vibe checks with photo/video upload, event selection, and description.

Dependencies:

- `scripts/components/modals.js`

Usage:

```javascript
import { loadComponent } from '../../scripts/utils/component-loader.js';
import { initializeVibeCheckModal, openModal } from '../../scripts/components/modals.js';

// Load and initialize the modal
loadComponent('modal-container', '../../components/modals/vibe-check-modal.html')
  .then(() => {
    initializeVibeCheckModal();
  });

// Open the modal
openModal('vibeCheckModal');
```

RSVP Modal

File: `components/modals/rsvp-modal.html`

Modal for RSVPing to events, with reminder settings.

Dependencies:

- `scripts/components/modals.js`

Usage:

```javascript
import { loadComponent } from '../../scripts/utils/component-loader.js';
import { openModal } from '../../scripts/components/modals.js';

// Load the modal
loadComponent('modal-container', '../../components/modals/rsvp-modal.html');

// Set event data in the modal
const modal = document.getElementById('rsvpModal');
modal.setAttribute('data-event-id', eventId);

// Open the modal
openModal('rsvpModal');
```

Edit Profile Modal

File: `components/modals/edit-profile-modal.html`

Modal for editing user profile information.

Dependencies:

- `scripts/components/profile.js`

Usage:

```javascript
import { loadComponent } from '../../scripts/utils/component-loader.js';
import { openModal } from '../../scripts/components/modals.js';

// Load the modal
loadComponent('modal-container', '../../components/modals/edit-profile-modal.html');

// Open the modal
openModal('editProfileModal');
```

Notification Settings Modal

File: `components/modals/notification-settings-modal.html`

Modal for managing notification preferences.

Dependencies:

- `scripts/services/notification-service.js`

Usage:

```javascript
import { loadComponent } from '../../scripts/utils/component-loader.js';
import { openModal } from '../../scripts/components/modals.js';

// Load the modal
loadComponent('modal-container', '../../components/modals/notification-settings-modal.html');

// Open the modal
openModal('notificationSettingsModal');
```

Recommended Events Component

File: `scripts/components/recommended-events.js`

Component that displays events recommended for the user.

Dependencies:

- `scripts/services/recommendation-service.js`

Usage:

```javascript
import RecommendedEventsComponent from '../../scripts/components/recommended-events.js';

// Initialize the component
const recommendedEvents = new RecommendedEventsComponent('recommended-container', {
  limit: 6,
  title: 'Recommended For You',
  showViewAll: true
});
```

Nearby Events Component

File: `scripts/components/nearby-events.js`

Component that displays events near the user's location.

Dependencies:

- `scripts/services/location-service.js`

Usage:

```javascript
import NearbyEventsComponent from '../../scripts/components/nearby-events.js';

// Initialize the component
const nearbyEvents = new NearbyEventsComponent('nearby-container', {
  radius: 10, // km
  limit: 6,
  showDistance: true
});
```

Implementation Plan

This detailed checklist provides a comprehensive plan for enhancing The Play with advanced features and improved user experience. The implementation follows the modular architecture established in Checklist 2, making it maintainable and extensible.

By completing these tasks, The Play will transform from a basic prototype into a fully-featured event discovery and social platform with:

1. A complete user profile system
2. Enhanced event details with RSVP functionality
3. Map and location features for finding nearby events
4. Advanced search and recommendations
5. Push notifications for event reminders

Next Steps

After implementing these features, future enhancements could include:

1. Chat and messaging system for event attendees
2. Event analytics for organizers
3. Integration with ticket payment providers
4. Enhanced AI recommendations
5. Community features for event groups and recurring events

Project Timeline

Based on the estimated durations, the complete implementation of Checklist 3 should take approximately 25 working days.

Sections 1-3 focus on core user experience improvements, while sections 4-5 provide more advanced features that enhance user engagement.