PECH Prayer Diary - Recommended Code Improvements

After reviewing the PECH Prayer Diary codebase, I've identified several areas where the code could be optimized or enhanced. Here are specific recommendations with example code implementations.

1. Performance Optimization for Calendar Loading

The current implementation of `loadPrayerCalendar()` in `calendar.js` makes separate queries for users and topics. We can optimize this with a single batch query.

javascript

```javascript
// Current implementation (two separate queries)
async function loadPrayerCalendar() {
    // ...
    const { data: userData, error: userError } = await supabase
        .from('profiles')
        .select(`id, full_name, photo_tag, profile_image_url, prayer_points, pray_day, pray_mor
        .eq('approval_state', 'Approved')
        .eq('pray_day', currentDay)
        .eq('calendar_hide', false)
        .or(`pray_months.eq.0,pray_months.eq.${isOddMonth ? 1 : 2}`)
        .order('full_name', { ascending: true});

    // ...

    const { data: topicData, error: topicError } = await supabase
        .from('prayer_topics')
        .select('*')
        .eq('pray_day', currentDay)
        .or(`pray_months.eq.0,pray_months.eq.${isOddMonth ? 1 : 2}`)
        .order('topic_title', { ascending: true });
    // ...
}

// Optimized implementation (single batch query)
async function loadPrayerCalendar() {
    // ...
    try {
        // Get current date or test date for determining which users to show
        const effectiveDate = getEffectiveDate();
        const currentDay = effectiveDate.getDate();
        const currentMonth = effectiveDate.getMonth() + 1;
        const isOddMonth = currentMonth % 2 === 1;

        // Update the date display
        // ...

        // Batch query both tables with Promise.all
        const [membersResponse, topicsResponse] = await Promise.all([
            supabase
                .from('profiles')
                .select(`id, full_name, photo_tag, profile_image_url, prayer_points, pray_day,
                .eq('approval_state', 'Approved')
                .eq('pray_day', currentDay)
                .eq('calendar_hide', false)
                .or(`pray_months.eq.0,pray_months.eq.${isOddMonth ? 1 : 2}`)
                .order('full_name', { ascending: true }),
            supabase
                .from('prayer_topics')
                .select('*')
                .eq('pray_day', currentDay)
                .or(`pray_months.eq.0,pray_months.eq.${isOddMonth ? 1 : 2}`)
                .order('topic_title', { ascending: true })
        ]);

        // Handle potential errors
        if (membersResponse.error) throw membersResponse.error;
        if (topicsResponse.error) throw topicsResponse.error;

        const userData = membersResponse.data || [];
        const topicData = topicsResponse.data || [];

        // Continue with the rest of the function...
    } catch (error) {
        // Error handling...
    }
}
```

2. Add Debounce for Search Functions

Several search functions could benefit from debouncing to prevent excessive queries when typing:

```javascript
// Add this utility function to the app
function debounce(func, wait) {
    let timeout;
    return function(...args) {
        const context = this;
        clearTimeout(timeout);
        timeout = setTimeout(() => func.apply(context, args), wait);
    };
}

// Then modify the event listeners in setupEventListeners() in calendar.js
function setupEventListeners() {
    // Search input with debounce
    const searchInput = document.getElementById('member-search');
    const debouncedFilter = debounce((value) => {
        filterAndDisplayUsers(value);
    }, 300); // 300ms debounce

    searchInput.addEventListener('input', () => {
        debouncedFilter(searchInput.value);
    });

    // Rest of event listeners...
}
```

3. Implement Caching for Prayer Updates

The app could benefit from client-side caching to reduce database queries:

javascript

```javascript
// Add to updates.js
const updatesCache = {
    data: null,
    timestamp: null,
    maxAge: 5 * 60 * 1000 // 5 minutes cache validity
};

async function loadPrayerUpdates() {
    if (!isApproved()) return;

    await window.waitForAuthStability();

    // Get container elements
    const latestContainer = document.getElementById('updates-container');
    const previousContainer = document.getElementById('archived-updates-container');

    // Show loading indicators
    latestContainer.innerHTML = createLoadingSpinner();
    previousContainer.innerHTML = createLoadingSpinner();

    try {
        let updates;

        // Check cache first
        const now = Date.now();
        if (updatesCache.data && updatesCache.timestamp && (now - updatesCache.timestamp < upda
            console.log('Using cached prayer updates data');
            updates = updatesCache.data;
        } else {
            // Load from database if cache is invalid
            const { data, error } = await supabase
                .from('prayer_updates')
                .select('*')
                .order('update_date', { ascending: false });

            if (error) throw error;

            // Update cache
            updates = data;
            updatesCache.data = updates;
            updatesCache.timestamp = now;
        }

        // Store updates for later reference
        window.allPrayerUpdates = updates || [];
        // Render updates...
    } catch (error) {
        // Error handling...
    }
}

// Add cache invalidation on create/update/delete
function invalidateUpdatesCache() {
    updatesCache.data = null;
    updatesCache.timestamp = null;
}

// Call invalidateUpdatesCache() in createPrayerUpdate() and deleteUpdate()
```

4. Enhance Error Handling with Retry Mechanism

For critical operations like saving updates, implement a retry mechanism:

javascript

```javascript
// Add to updates.js
async function withRetry(operation, maxRetries = 3) {
    let retries = 0;

    while (retries < maxRetries) {
        try {
            return await operation();
        } catch (error) {
            retries++;
            console.warn(`Operation failed (attempt ${retries}/${maxRetries}):`, error);

            if (retries >= maxRetries) {
                throw error; // Max retries reached, propagate the error
            }

            // Exponential backoff
            const delay = Math.min(1000 * (2 ** (retries - 1)) + Math.random() * 1000, 10000);
            await new Promise(resolve => setTimeout(resolve, delay));
        }
    }
}

// Then use it in createPrayerUpdate
async function createPrayerUpdate(action, submitBtn) {
    // ...preparing data...

    try {
        if (isEditing) {
            // Update with retry
            const { data, error } = await withRetry(() =>
                supabase
                    .from('prayer_updates')
                    .update({
                        title,
                        content,
                        update_date: dateInput
                    })
                    .eq('id', selectedUpdateId)
            );

            if (error) throw error;
        } else {
            // Create with retry
            const { data, error } = await withRetry(() =>
                supabase
                    .from('prayer_updates')
                    .insert({
                        title,
                        content,
                        created_by: userId,
                        is_archived: false,
                        update_date: dateInput
                    })
            );

            if (error) throw error;
        }

        // ...rest of function...
    } catch (error) {
        // ...error handling...
    }
}
```

5. Add Offline Indicator and Queue

Since this is a PWA, add offline support with a queuing system for operations:

javascript

```javascript
// Add to a new file called offline.js
const operationQueue = [];
let isOnline = navigator.onLine;

// Update online status
window.addEventListener('online', () => {
    isOnline = true;
    document.body.classList.remove('offline-mode');
    document.getElementById('offline-indicator')?.classList.add('d-none');

    // Process queued operations
    processQueue();
});

window.addEventListener('offline', () => {
    isOnline = false;
    document.body.classList.add('offline-mode');
    document.getElementById('offline-indicator')?.classList.remove('d-none');
});

// Queue operation
function queueOperation(type, data) {
    const operation = {
        id: Date.now().toString(),
        type,
        data,
        timestamp: Date.now()
    };

    operationQueue.push(operation);
    saveQueue();

    return operation.id;
}

// Save queue to localStorage
function saveQueue() {
    try {
        localStorage.setItem('operationQueue', JSON.stringify(operationQueue));
    } catch (e) {
        console.error('Could not save operation queue:', e);
    }
}

// Load queue from localStorage
function loadQueue() {
        try {
            const queueData = localStorage.getItem('operationQueue');
            if (queueData) {
                const parsedQueue = JSON.parse(queueData);
                operationQueue.length = 0;
                operationQueue.push(...parsedQueue);
            }
        } catch (e) {
            console.error('Could not load operation queue:', e);
        }
    }

    // Process the operation queue
    async function processQueue() {
        if (!isOnline || operationQueue.length === 0) return;

        // Process one operation at a time
        const operation = operationQueue[0];

        try {
            // Process based on operation type
            switch (operation.type) {
                case 'createUpdate':
                    await processCreateUpdate(operation.data);
                    break;
                case 'deleteUpdate':
                    await processDeleteUpdate(operation.data);
                    break;
                // Add other operation types as needed
            }

            // Operation successful, remove from queue
            operationQueue.shift();
            saveQueue();

            // Process next operation if any
            if (operationQueue.length > 0) {
                processQueue();
            }
        } catch (error) {
            console.error('Error processing queued operation:', error);

            // If offline again, stop processing
            if (!isOnline) return;

            // If online but failed, retry after delay
            setTimeout(processQueue, 60000); // 1 minute
    }
}

// Add to index.html
// <div id="offline-indicator" class="d-none position-fixed bottom-0 start-0 mb-3 ms-3 p-2 bg-w
//     <i class="bi bi-wifi-off me-2"></i> Offline Mode
// </div>
```

6. Optimize User Permissions Check

The current permission checks are repeated in many places. Create a centralized permission system:

```
javascript
```

```javascript
// Add to a new file called permissions.js
const userPermissions = {
    // Cache for user profile
    profile: null,

    // Get the user profile with caching
    async getProfile() {
        if (this.profile && this.profile.id === getUserId()) {
            return this.profile;
        }

        // Fetch profile from database
        try {
            const { data, error } = await supabase
                .from('profiles')
                .select('*')
                .eq('id', getUserId())
                .single();

            if (error) throw error;

            this.profile = data;
            return data;
        } catch (error) {
            console.error('Error fetching user profile:', error);
            return null;
        }
    },

    // Clear the cache
    clearCache() {
        this.profile = null;
    },

    // Check if user has a permission
    async can(permission) {
        const profile = await this.getProfile();
        if (!profile) return false;

        // Admins have all permissions
        if (profile.user_role === 'Administrator') return true;

        // Check specific permission
        switch(permission) {
            case 'edit_calendar':
                return !!profile.prayer_calendar_editor;
            case 'edit_updates':
                return !!profile.prayer_update_editor;
            case 'edit_urgent':
                return !!profile.urgent_prayer_editor;
            default:
                return false;
        }
    }
};

// Then use it like this in any file:
// if (await userPermissions.can('edit_calendar')) {
//     // Do calendar editing stuff
// }
```

7. Standardize Date Handling

The app uses different date formatting in different places. Let's standardize it:

javascript

```javascript
// Add to a new file called date-utils.js
const dateUtils = {
    // Format date as YYYY-MM-DD for inputs
    formatForInput(date) {
        if (!date) date = new Date();
        return date.toISOString().split('T')[0];
    },

    // Format date as "24 Apr 2025" (short display format)
    formatShort(date) {
        if (!date) date = new Date();
        const options = { day: 'numeric', month: 'short', year: 'numeric' };
        return date.toLocaleDateString(undefined, options);
    },

    // Format date as "24 April 2025" (long display format)
    formatLong(date) {
        if (!date) date = new Date();
        const options = { day: 'numeric', month: 'long', year: 'numeric' };
        return date.toLocaleDateString(undefined, options);
    },

    // Parse a date string in any common format
    parse(dateString) {
        if (!dateString) return new Date();

        // Try different formats
        if (dateString.match(/^\d{4}-\d{2}-\d{2}$/)) {
            // YYYY-MM-DD format
            const [year, month, day] = dateString.split('-').map(Number);
            return new Date(year, month - 1, day);
        }

        // Default: let JavaScript handle it
        return new Date(dateString);
    },

    // Get the day of month (1-31)
    getDayOfMonth(date) {
        if (!date) date = new Date();
        return date.getDate();
    },

    // Check if month is odd or even
    isOddMonth(date) {
        if (!date) date = new Date();
        return (date.getMonth() + 1) % 2 === 1;
    }
};

// Then replace all date formatting with these standardized methods
```

8. Lazy Loading Editors

To improve initial load time, lazy load the Quill editor only when needed:

javascript

```javascript
let quillPromise = null;

// Lazy load Quill
function loadQuill() {
    if (!quillPromise) {
        quillPromise = new Promise((resolve) => {
            // Check if Quill is already loaded
            if (window.Quill) {
                resolve(window.Quill);
                return;
            }

            // Load Quill script
            const script = document.createElement('script');
            script.src = 'https://cdn.jsdelivr.net/npm/quill@2.0.3/dist/quill.min.js';
            script.onload = () => resolve(window.Quill);
            document.head.appendChild(script);
        });
    }

    return quillPromise;
}

// Modified initUpdateEditor to use lazy loading
async function initUpdateEditor() {
    console.log('DEBUG: initUpdateEditor - Start initialization');
    if (initUpdateEditorFlag) {
        console.log('DEBUG: initUpdateEditor - Duplicate call detected, aborting');
        return;
    }

    // Load Quill on demand
    console.log('DEBUG: initUpdateEditor - Loading Quill editor');
    const Quill = await loadQuill();

    // Define custom formats that exclude direct color styling
    const allowedFormats = [
        'bold', 'italic', 'underline', 'strike',
        'header', 'list', 'bullet', 'indent',
        'link', 'image', 'direction', 'align', 'blockquote'
    ];

    // Continue initializing as before...
}
```

9. Improve Service Worker Update Detection

Enhance the way the app detects and handles service worker updates:

javascript

```
// Add to service-worker.js
const APP_VERSION = '1.1.093'; // Must match config.js version

self.addEventListener('message', (event) => {
    if (event.data && event.data.action === 'CHECK_FOR_UPDATES') {
        const clientVersion = event.data.version;

        // Compare versions and notify if different
        if (clientVersion !== APP_VERSION) {
            self.clients.matchAll().then(clients => {
                clients.forEach(client => {
                    client.postMessage({
                        type: 'UPDATE_AVAILABLE',
                        currentVersion: APP_VERSION
                    });
                });
            });
        }
    }
    // Other event handlers...
});

// Add to app.js for more reliable checking
function checkForAppUpdate(registration) {
    console.log('Checking for app updates...');

    // First check against service worker version
    if (navigator.serviceWorker.controller) {
        navigator.serviceWorker.controller.postMessage({
            action: 'CHECK_FOR_UPDATES',
            version: APP_VERSION
        });
    }

    // Also check for new service worker registration
    if (registration.waiting) {
        console.log('New service worker waiting');
        // Show update notification
        showUpdateNotification('New Version');
    }

    // Listen for updates to the current service worker
    registration.addEventListener('updatefound', () => {
        const newWorker = registration.installing;

        newWorker.addEventListener('statechange', () => {
            if (newWorker.state === 'installed' && navigator.serviceWorker.controller) {
                console.log('New service worker installed');
                // Show update notification
                showUpdateNotification('New Version');
            }
        });
    });
}
```

10. Add Accessibility Enhancements

Improve the accessibility of the app for users with disabilities:

```javascript
// Add to app.js
function enhanceAccessibility() {
    // Add appropriate ARIA attributes to interactive elements
    document.querySelectorAll('.prayer-card').forEach(card => {
        card.setAttribute('role', 'article');
        card.setAttribute('tabindex', '0'); // Make focusable
    });

    // Make sure all img elements have alt text
    document.querySelectorAll('img:not([alt])').forEach(img => {
        img.setAttribute('alt', 'Prayer diary image');
    });

    // Ensure form elements have labels
    document.querySelectorAll('input, select, textarea').forEach(formElement => {
        const id = formElement.getAttribute('id');
        if (id) {
            const hasLabel = document.querySelector(`label[for="${id}"]`);
            if (!hasLabel) {
                console.warn(`Form element with ID ${id} is missing a label`);
            }
        }
    });

    // Ensure color contrast for dark mode
    document.querySelectorAll('.content-container *').forEach(element => {
        // Remove any inline color styles that might reduce contrast
        if (element.style && element.style.color) {
            element.style.color = '';
        }
    });
}

// Call this function after loading views
document.addEventListener('view-shown', enhanceAccessibility);
```

These improvements should help optimize performance, enhance user experience, and improve the overall code quality of your PECH Prayer Diary app.