Gmail Functionality Preservation Guide

Original Gmail Implementation (script.js)

The original script.js implements Gmail functionality with these key components:

1. **OAuth Flow**:
   - `initiateGmailAuth()`: Begins the OAuth process
   - `generateRandomState()`: Creates a random state for OAuth security
   - `handleOAuthCallback()`: Processes the OAuth response
   - `checkForOAuthCallback()`: Checks for OAuth callbacks on load
2. **API Initialization**:
   - `gapiLoaded()`: Initializes GAPI client
   - `gisLoaded()`: Initializes Google Identity Services
   - `loadGmailApi()`: Loads the Gmail API
3. **Email Processing**:
   - `startReadingEmails()`: Begins reading emails
   - `readEmails()`: Fetches emails from Gmail API
   - `getEmailDetails()`: Gets details for a specific email
   - `readEmailsOneByOne()`: Reads emails with voice prompts
   - `waitForNextCommandWithTimeout()`: Waits for voice command with timeout
4. **State Management**:
   - `handleGmailAuthSuccess()`: Handles successful authentication
   - `handleGmailAuthFailure()`: Handles failed authentication
   - `handleGmailCommands()`: Processes Gmail voice commands
   - `startGmailCommandLoop()`: Starts a loop to listen for Gmail commands
   - `handleGmailSignout()`: Signs out of Gmail

Refactored GmailManager.js

The refactored GmailManager.js implements much of this functionality but there are some areas to address:

```javascript
// Example of refactored GmailManager (simplified)
export class GmailManager {
    constructor(websocketManager, speechManager) {
        this.websocket = websocketManager;
        this.speech = speechManager;
        this.tokenClient = null;
        this.gapiInited = false;
        this.gisInited = false;
        this.gmailCommandAttempts = 0;
        this.MAX_GMAIL_COMMAND_ATTEMPTS = 3;
        this.authHandled = false;
    }

    // Methods...
}
```

Gaps and Issues to Address

1. Missing State Management

The GmailManager should handle state transitions that were previously managed by the wakeWordState variable. Add a proper state management system:

```javascript
// Add to GmailManager
this.state = {
    isAuthorized: false,
    isReading: false,
    currentEmailIndex: 0,
    totalEmails: 0,
    commandMode: 'inactive' // 'inactive', 'reading', 'awaiting_command'
};

updateState(newState) {
    this.state = { ...this.state, ...newState };
    console.log('Gmail state updated:', this.state);
}
```

2. Improved Email Reading Loop

The original implementation uses multiple nested functions for email reading. Refactor for clarity:

javascript

```
async readEmailsOneByOne(emails) {
    this.updateState({
        isReading: true,
        currentEmailIndex: 0,
        totalEmails: emails.length
    });

    while (this.state.currentEmailIndex < this.state.totalEmails) {
        const email = emails[this.state.currentEmailIndex];

        // Announce current email
        await this.announceCurrentEmail(email);

        // Process command for this email
        const command = await this.getNextEmailCommand();

        if (command === "next") {
            this.updateState({ currentEmailIndex: this.state.currentEmailIndex + 1 });
        } else if (command === "finish") {
            break;
        }
    }

    await this.finishEmailReading();
}

async announceCurrentEmail(email) {
    const emailContent = `Email ${this.state.currentEmailIndex + 1} of ${this.state.totalEmails
                          From ${email.from}: Subject: ${email.subject}`;
    return this.speech.speakFeedback(emailContent);
}

async getNextEmailCommand() {
    this.updateState({ commandMode: 'awaiting_command' });

    await this.speech.speakFeedback("Say 'next' for the next email or 'finish' to stop.");

    try {
        const command = await this.waitForCommand(20000);

        if (command === "timeout") {
            await this.speech.speakFeedback("No command received. Please try again.");
            return await this.getNextEmailCommand();
```

```
        } else if (command.includes("finish")) {
            return "finish";
        } else if (command.includes("next")) {
            return "next";
        } else {
            await this.speech.speakFeedback("Command not recognized. Please say 'next' or 'fini
            return await this.getNextEmailCommand();
        }
    } catch (error) {
        console.error("Error waiting for command:", error);
        await this.speech.speakFeedback("Error processing command. Please try again.");
        return await this.getNextEmailCommand();
    }
}
```

3. OAuth Flow Improvements

The OAuth flow needs careful handling to maintain security and reliability:

javascript

```
async initiateGmailAuth() {
    console.log("Starting Gmail authentication process");
    const accessToken = localStorage.getItem('gmail_access_token');

    if (!accessToken) {
        console.log("No access token found, initiating OAuth flow");

        // Use CONFIG for client ID and redirect URI
        const clientId = CONFIG.GMAIL.CLIENT_ID;
        const redirectUri = encodeURIComponent(CONFIG.GMAIL.REDIRECT_URI);
        const scope = encodeURIComponent(CONFIG.GMAIL.SCOPES);
        const state = encodeURIComponent(this.generateRandomState());

        const authUrl = `https://accounts.google.com/o/oauth2/v2/auth?` +
            `client_id=${clientId}&` +
            `redirect_uri=${redirectUri}&` +
            `response_type=token&` +
            `scope=${scope}&` +
            `state=${state}&` +
            `include_granted_scopes=true`;

        // Show the auth prompt UI element
        document.getElementById('gmailAuthPrompt').style.display = 'block';

        const authWindow = window.open(authUrl, 'Gmail Authorization', 'width=600,height=600');

        if (authWindow) {
            this.setupAuthMessageListener(authWindow);
        } else {
            console.error("Could not open authorization window");
            await this.speech.speakFeedback("Could not open Gmail authorization window. Please
            document.getElementById('gmailAuthPrompt').style.display = 'none';
        }
    } else {
        console.log("Using existing access token");
        await this.handleGmailAuthSuccess();
    }
}

setupAuthMessageListener(authWindow) {
    window.addEventListener('message', async (event) => {
        // Only accept messages from our redirect URI origin
        const redirectOrigin = new URL(CONFIG.GMAIL.REDIRECT_URI).origin;
```

```
        if (event.origin !== redirectOrigin) {
            console.warn("Unexpected origin for OAuth callback:", event.origin);
            return;
        }

        if (event.data.type === 'OAUTH_CALLBACK') {
            console.log("Received OAuth callback");
            document.getElementById('gmailAuthPrompt').style.display = 'none';

            if (event.data.accessToken) {
                localStorage.setItem('gmail_access_token', event.data.accessToken);
                await this.handleGmailAuthSuccess();
            } else {
                await this.handleGmailAuthFailure("No access token received");
            }
        }

        if (event.data.type === 'OAUTH_CLOSE_WINDOW') {
            authWindow.close();
        }
    }, false);
}
```

4. Error Handling and Recovery

Add comprehensive error handling for Gmail operations:

javascript

```
async loadGmailApi() {
    return new Promise((resolve, reject) => {
        try {
            if (!this.gapiInited) {
                gapi.load('client', async () => {
                    try {
                        await gapi.client.init({
                            apiKey: CONFIG.GMAIL.API_KEY,
                            discoveryDocs: [CONFIG.GMAIL.DISCOVERY_DOC],
                        });
                        console.log("Gmail API initialized and loaded");
                        this.gapiInited = true;
                        resolve();
                    } catch (error) {
                        console.error("Error initializing Gmail API:", error);
                        reject(error);
                    }
                });
            } else {
                gapi.client.load('gmail', 'v1', () => {
                    console.log("Gmail API loaded");
                    resolve();
                });
            }
        } catch (error) {
            console.error("Critical error loading Gmail API:", error);
            reject(error);
        }
    });
}

async getEmailDetails(messageId) {
    let retryCount = 0;
    const maxRetries = 3;

    while (retryCount < maxRetries) {
        try {
            const response = await gapi.client.gmail.users.messages.get({
                'userId': 'me',
                'id': messageId
            });

            const message = response.result;
```

```
            const headers = message.payload.headers;
            const subject = headers.find(header => header.name === "Subject")?.value || "No sub
            const from = headers.find(header => header.name === "From")?.value || "Unknown send

            return { subject, from };
        } catch (err) {
            console.error(`Error getting email details (attempt ${retryCount + 1}):`, err);
            retryCount++;

            if (retryCount < maxRetries) {
                // Exponential backoff
                await new Promise(resolve => setTimeout(resolve, 1000 * Math.pow(2, retryCount)
            } else {
                return {
                    subject: 'Error retrieving subject',
                    from: 'Error retrieving sender',
                    error: err
                };
            }
        }
    }
}
```

5. Ensuring Clean Command Transitions

Add clean transition methods for moving between states:

```javascript
async cycleToMainMenu() {
    console.log("Cycling to main menu from Gmail");
    this.updateState({
        isReading: false,
        commandMode: 'inactive'
    });

    await this.speech.speakFeedback("Email reading finished. Returning to main menu.");

    // Signal to SpeechManager to return to listening state
    if (this.speech) {
        this.speech.wakeWordState = 'listening';
        await this.speech.cycleToMainMenu();
    }
}

async handleGmailSignout() {
    localStorage.removeItem('gmail_access_token');
    this.updateState({
        isAuthorized: false,
        isReading: false,
        commandMode: 'inactive'
    });

    await this.speech.speakFeedback("Signed out of Gmail.");

    // Return to main menu
    if (this.speech) {
        this.speech.wakeWordState = 'listening';
        await this.speech.cycleToMainMenu();
    }
}
```

Integration with App.js

Ensure proper initialization in App.js:

```javascript
// In App.js constructor
this.gmail = new GmailManager(this.websocket, this.speech);

// Connect speech to gmail for transitions
this.speech.gmail = this.gmail;

// Initialize Gmail API if scripts are loaded
if (typeof gapi !== 'undefined') {
    this.gmail.checkForOAuthCallback();
}
```

Required HTML Changes

Add needed HTML elements for Gmail integration:

```html
<!-- Add to index.html before closing body tag -->
<script async defer src="https://apis.google.com/js/api.js" onload="window.gapiLoaded && windo
<script async defer src="https://accounts.google.com/gsi/client" onload="window.gisLoaded && wi

<div id="gmailAuthPrompt" style="display: none; background-color: yellow; padding: 10px; margir
    Please check your browser windows for a Google authorization prompt.
    Complete the authorization process, then say "retry Gmail" to continue.
</div>
```

Connection with SpeechManager

The Gmail commands need to be properly handled by SpeechManager:

javascript

```javascript
// Add this to SpeechManager's handleMenuCommand method
else if (command.includes("gmail")) {
    if (window.rentAHal.gmail) {
        await this.speakFeedback("Accessing Gmail...");
        this.wakeWordState = 'gmail';
        await window.rentAHal.gmail.initiateGmailAuth();
    } else {
        await this.speakFeedback("Gmail service is not available at the moment.");
        await this.cycleToMainMenu();
    }
}
```

Service Worker Integration

If you're using a service worker, ensure it doesn't cache the OAuth redirect:

javascript

```javascript
// In service-worker.js
self.addEventListener('fetch', (event) => {
    // Don't cache OAuth redirects or API calls
    if (event.request.url.includes('oauth') ||
        event.request.url.includes('googleapis.com')) {
        return;
    }
    // Handle other fetches
    // ...
});
```

Testing Gmail Integration

1. **OAuth Flow Test**:
   - Clear localStorage and test complete flow
   - Verify token storage
   - Test refreshing token
2. **Email Reading Test**:
   - Test with various email counts
   - Verify proper reading of emails
   - Test command recognition

3. **Error Recovery Test**:
    - Test with network disconnection
    - Test with invalid tokens
    - Test with API errors
4. **Voice Command Test**:
    - Test all Gmail voice commands
    - Verify command recognition accuracy
    - Test transitions between modes

By following these guidelines, the Gmail functionality should be successfully preserved during the refactoring process.