AlertService.gs
CoachingService.gs
Code.gs
DataService.gs
ManagerService.gs
MetricsService.gs
OneOnOneService.gs
QualityService.gs
ScheduleService.gs
SettingsService.gs
ShiftService.gs
TaskService.gs
TeamService.gs
TimeTracker.gs
UIService.gs
ValidationSystem.gs
active-alerts.html
add-counselor-form.html
alert-form.html
asana-task-form.html
coaching-form.html
crisis-protocols.html
dashboard.html
help-resources.html
initialize-week-form.html
manager-one-on-ones.html
metrics-form.html
metrics-report.html
one-on-one.html
quality-reports.html
quality-review-form.html
schedule-manager.html
settings.html
shift-initialization.html
sidebar.html
task-form.html
time-tracker.html
update-status-form.html

```
/**
* Service for managing alerts
*/
const AlertService = {
 /**
  * Creates a new alert
  * @param {object} alertData - Data for the new alert
  * @return {object} Result of the operation
  */
 createAlert: function(alertData) {
   try {
     const ss = SpreadsheetApp.getActiveSpreadsheet();
     let sheet = ss.getSheetByName(CRISIS_SUPPORT_CONFIG.SHEETS.ALERTS);

     // Create sheet if it doesn't exist
     if (!sheet) {
       sheet = ss.insertSheet(CRISIS_SUPPORT_CONFIG.SHEETS.ALERTS);
       sheet.appendRow([
         'ID',
         'Date',
         'Type',
         'Message',
         'Priority',
         'Assigned To',
         'Created By',
         'Expiration Date',
         'Status',
         'Resolution Notes',
         'Resolved By',
         'Resolution Date',
         'Snoozed Until'
       ]);
       sheet.getRange(1, 1, 1, 13)
```

```
        .setFontWeight('bold')
        .setBackground('#E0E0E0');
    }

    // Generate unique ID
    const alertId = Utilities.getUuid();

    // Get current user
    const currentUser = Session.getActiveUser().getEmail();

    // Prepare row data
    const rowData = [
      alertId,
      new Date(),
      alertData.type,
      alertData.message,
      alertData.priority,
      alertData.assignedTo || '',
      currentUser,
      alertData.expirationDate ? new Date(alertData.expirationDate) : '',
      'Active',
      '',
      '',
      '',
      ''
    ];

    // Add to sheet
    sheet.appendRow(rowData);

    // Log activity
    logSystemActivity('Alert Management', `Created new ${alertData.priority}
priority alert: ${alertData.type}`);
```

```
    return {
      success: true,
      message: 'Alert created successfully',
      alertId: alertId
    };
  } catch (error) {
    logError('createAlert', error);
    return {
      success: false,
      message: 'Error creating alert: ' + error.message
    };
  }
},
 /**
 * Gets all active alerts
 * @return {Array} Active alerts
 */
getActiveAlerts: function() {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    const sheet = ss.getSheetByName(CRISIS_SUPPORT_CONFIG.SHEETS.ALERTS);

    // Return empty array if sheet doesn't exist
    if (!sheet) {
      return [];
    }

    // Get all data
    const data = sheet.getDataRange().getValues();

    // Skip if only header row
    if (data.length <= 1) {
```

```
      return [];
    }

    // Get column indices
    const headers = data[0];
    const idIndex = headers.indexOf('ID');
    const dateIndex = headers.indexOf('Date');
    const typeIndex = headers.indexOf('Type');
    const messageIndex = headers.indexOf('Message');
    const priorityIndex = headers.indexOf('Priority');
    const assignedToIndex = headers.indexOf('Assigned To');
    const createdByIndex = headers.indexOf('Created By');
    const expirationDateIndex = headers.indexOf('Expiration Date');
    const statusIndex = headers.indexOf('Status');
    const snoozedUntilIndex = headers.indexOf('Snoozed Until');

    // Get current time
    const now = new Date();

    // Filter active alerts
    const activeAlerts = [];

    for (let i = 1; i < data.length; i++) {
      // Skip resolved or closed alerts
      if (data[i][statusIndex] === 'Resolved' || data[i][statusIndex] ===
'Closed') {
        continue;
      }

      // Check if alert is expired
      const expirationDate = data[i][expirationDateIndex];
      if (expirationDate && expirationDate < now) {
        continue;
```

```
      }

      // Check if alert is snoozed
      const snoozedUntil = data[i][snoozedUntilIndex];
      if (snoozedUntil && snoozedUntil > now) {
        continue;
      }

      // Add to active alerts
      activeAlerts.push({
        id: data[i][idIndex],
        date: data[i][dateIndex],
        type: data[i][typeIndex],
        message: data[i][messageIndex],
        priority: data[i][priorityIndex],
        assignedTo: data[i][assignedToIndex],
        createdBy: data[i][createdByIndex],
        expirationDate: data[i][expirationDateIndex],
        status: data[i][statusIndex]
      });
    }

    return activeAlerts;
  } catch (error) {
    logError('getActiveAlerts', error);
    return [];
  }
},
 /**
 * Snoozes an alert for 24 hours
 * @param {string} alertId - ID of the alert to snooze
 * @return {object} Result of the operation
 */
```

```
snoozeAlert: function(alertId) {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    const sheet = ss.getSheetByName(CRISIS_SUPPORT_CONFIG.SHEETS.ALERTS);

    // Return error if sheet doesn't exist
    if (!sheet) {
      return {
        success: false,
        message: 'Alerts sheet not found'
      };
    }

    // Get all data
    const data = sheet.getDataRange().getValues();

    // Get column indices
    const headers = data[0];
    const idIndex = headers.indexOf('ID');
    const snoozedUntilIndex = headers.indexOf('Snoozed Until');

    // Find the alert
    let alertRow = -1;

    for (let i = 1; i < data.length; i++) {
      if (data[i][idIndex] === alertId) {
        alertRow = i + 1; // +1 because sheet rows are 1-indexed
        break;
      }
    }

    // Return error if alert not found
    if (alertRow === -1) {
```

```
      return {
        success: false,
        message: 'Alert not found'
      };
    }

    // Calculate snooze time (24 hours from now)
    const snoozeUntil = new Date();
    snoozeUntil.setHours(snoozeUntil.getHours() + 24);

    // Update the sheet
    sheet.getRange(alertRow, snoozedUntilIndex + 1).setValue(snoozeUntil);

    // Log activity
    logSystemActivity('Alert Management', `Snoozed alert ${alertId} for 24
hours`);

    return {
      success: true,
      message: 'Alert snoozed for 24 hours'
    };
  } catch (error) {
    logError('snoozeAlert', error);
    return {
      success: false,
      message: 'Error snoozing alert: ' + error.message
    };
  }
},
 /**
 * Resolves an alert
 * @param {string} alertId - ID of the alert to resolve
 * @param {string} resolutionNotes - Notes about the resolution
```

```
 * @return {object} Result of the operation
 */
resolveAlert: function(alertId, resolutionNotes) {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    const sheet = ss.getSheetByName(CRISIS_SUPPORT_CONFIG.SHEETS.ALERTS);

    // Return error if sheet doesn't exist
    if (!sheet) {
      return {
        success: false,
        message: 'Alerts sheet not found'
      };
    }

    // Get all data
    const data = sheet.getDataRange().getValues();

    // Get column indices
    const headers = data[0];
    const idIndex = headers.indexOf('ID');
    const statusIndex = headers.indexOf('Status');
    const resolutionNotesIndex = headers.indexOf('Resolution Notes');
    const resolvedByIndex = headers.indexOf('Resolved By');
    const resolutionDateIndex = headers.indexOf('Resolution Date');

    // Find the alert
    let alertRow = -1;

    for (let i = 1; i < data.length; i++) {
      if (data[i][idIndex] === alertId) {
        alertRow = i + 1; // +1 because sheet rows are 1-indexed
        break;
```

```
      }
    }

    // Return error if alert not found
    if (alertRow === -1) {
      return {
        success: false,
        message: 'Alert not found'
      };
    }

    // Get current user
    const currentUser = Session.getActiveUser().getEmail();

    // Get current date
    const now = new Date();

    // Update the sheet
    sheet.getRange(alertRow, statusIndex + 1).setValue('Resolved');
    sheet.getRange(alertRow, resolutionNotesIndex +
1).setValue(resolutionNotes);
    sheet.getRange(alertRow, resolvedByIndex + 1).setValue(currentUser);
    sheet.getRange(alertRow, resolutionDateIndex + 1).setValue(now);

    // Log activity
    logSystemActivity('Alert Management', `Resolved alert ${alertId}`);

    return {
      success: true,
      message: 'Alert resolved successfully'
    };
  } catch (error) {
    logError('resolveAlert', error);
```

```
      return {
        success: false,
        message: 'Error resolving alert: ' + error.message
      };
    }
  }
};

// Expose functions to UI
function createAlert(alertData) {
  return AlertService.createAlert(alertData);
}

function getActiveAlerts() {
  return AlertService.getActiveAlerts();
}

function snoozeAlert(alertId) {
  return AlertService.snoozeAlert(alertId);
}

function resolveAlert(alertId, resolutionNotes) {
  return AlertService.resolveAlert(alertId, resolutionNotes);
}
/**
* CoachingService.gs
* Service for managing coaching notes in the Crisis Support System
*/
const CoachingService = {
  /**
   * Get coaching notes for a specific counselor
   * @param {string} counselorEmail - Email of the counselor
   * @return {Array} Array of coaching note objects
```

```
 */
getCoachingNotes: function(counselorEmail) {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    let sheet = ss.getSheetByName('Coaching Notes');

    if (!sheet) {
      // Return empty array if sheet doesn't exist
      return [];
    }

    // Get all data
    const data = sheet.getDataRange().getValues();

    // Skip header row
    if (data.length <= 1) {
      return [];
    }

    // Get column indices from headers
    const headers = data[0];
    const idIndex = headers.indexOf('ID');
    const dateIndex = headers.indexOf('Date');
    const emailIndex = headers.indexOf('Email');
    const counselorIndex = headers.indexOf('Counselor');
    const coachIndex = headers.indexOf('Coach');
    const typeIndex = headers.indexOf('Type');
    const focusIndex = headers.indexOf('Focus');
    const summaryIndex = headers.indexOf('Summary');
    const followUpDateIndex = headers.indexOf('Follow-Up Date');

    // Check if required columns exist
    if (dateIndex === -1 || emailIndex === -1) {
```

```
      console.error('Required columns not found in Coaching Notes sheet');
      return [];
    }

    // Map data to coaching notes
    const notes = [];

    for (let i = 1; i < data.length; i++) {
      const row = data[i];

      // Filter by counselor email
      if (row[emailIndex] && row[emailIndex].toLowerCase() ===
counselorEmail.toLowerCase()) {
        notes.push({
          id: idIndex !== -1 ? row[idIndex] : `note_${i}`,
          date: dateIndex !== -1 ? row[dateIndex] : null,
          counselor: counselorIndex !== -1 ? row[counselorIndex] : '',
          email: row[emailIndex],
          coach: coachIndex !== -1 ? row[coachIndex] : '',
          type: typeIndex !== -1 ? row[typeIndex] : '',
          focus: focusIndex !== -1 ? row[focusIndex] : '',
          summary: summaryIndex !== -1 ? row[summaryIndex] : '',
          followUpDate: followUpDateIndex !== -1 ? row[followUpDateIndex] :
null
        });
      }
    }

    return notes;
  } catch (error) {
    logError('CoachingService.getCoachingNotes', error);
    return [];
  }
```

```
},
 /**
 * Get all counselors from the Counselor Tracking sheet
 * @return {Array} Array of counselor objects
 */
getAllCounselors: function() {
  // Delegate to TeamService to ensure single point of truth
  return TeamService.getTeamMembers();
},
 /**
 * Get details for a specific coaching note
 * @param {string} noteId - ID of the coaching note
 * @return {Object} Coaching note details or null if not found
 */
getCoachingNoteDetails: function(noteId) {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    const sheet = ss.getSheetByName('Coaching Notes');

    if (!sheet) {
      return null;
    }

    // Get all data
    const data = sheet.getDataRange().getValues();

    // Get column indices from headers
    const headers = data[0];
    const idIndex = headers.indexOf('ID');

    // Find coaching note by ID
    for (let i = 1; i < data.length; i++) {
      const row = data[i];
```

```
      if (idIndex !== -1 && row[idIndex] === noteId) {
        // Create coaching note object with all properties
        const note = {
          id: row[idIndex]
        };

        // Add all other fields
        for (let j = 0; j < headers.length; j++) {
          if (j !== idIndex) {
            note[this.camelCase(headers[j])] = row[j];
          }
        }

        return note;
      }
    }

    return null;
  } catch (error) {
    logError('CoachingService.getCoachingNoteDetails', error);
    return null;
  }
},
 /**
 * Save a new coaching note
 * @param {Object} noteData - Data for the coaching note
 * @return {Object} Result of the operation
 */
saveCoachingNote: function(noteData) {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    let sheet = ss.getSheetByName('Coaching Notes');
```

```
// Create sheet if it doesn't exist
if (!sheet) {
  sheet = ss.insertSheet('Coaching Notes');
  sheet.appendRow([
    'ID',
    'Date',
    'Counselor',
    'Email',
    'Coach',
    'Type',
    'Focus',
    'Summary',
    'Follow-Up Date',
    'Prep Gather Concerns',
    'Prep Review Documentation',
    'Prep Collect Evidence',
    'Prep Identify Root Causes',
    'Concern Areas',
    'Prep Verify Documentation',
    'Prep Prepare Examples',
    'Prep Identify Resources',
    'Prep Develop Approach',
    'Preparation Reflection',
    'Initial Connection',
    'Concern Exploration',
    'Development Planning',
    'Doc Summarize Points',
    'Doc Agreed Actions',
    'Doc Clarify Expectations',
    'Doc Establish Timeline',
    'Formal Intervention',
    'Action Items',
```

```
        'Confidential Notes',
        'Created By',
        'Created Date'
      ]);
      sheet.getRange(1, 1, 1, 31)
        .setFontWeight('bold')
        .setBackground('#E0E0E0');
    }

    // Convert noteData to proper format
    const formattedData = this.formatCoachingData(noteData);

    // Get current user info
    const currentUser = Session.getActiveUser().getEmail();

    // Generate a unique ID for the coaching note
    const noteId = Utilities.getUuid();

    // Prepare row data
    const rowData = [
      noteId,                                          // ID
      new Date(noteData.date),                         // Date
      noteData.counselorName,                          // Counselor
      noteData.counselorEmail,                         // Email
      currentUser,                                     // Coach
      noteData.type || 'Regular',                      // Type
      noteData.focus || '',                            // Focus
      noteData.summary || '',                          // Summary
      noteData.followUpDate ? new Date(noteData.followUpDate) : '', //
Follow-Up Date
      formattedData.prepGatherConcerns || false,   // Prep Gather Concerns
      formattedData.prepReviewDocumentation || false, // Prep Review
Documentation
```

```
      formattedData.prepCollectEvidence || false,  // Prep Collect Evidence
      formattedData.prepIdentifyRootCauses || false, // Prep Identify Root
Causes
      formattedData.concernAreas || '',            // Concern Areas
      formattedData.prepVerifyDocumentation || false, // Prep Verify
Documentation
      formattedData.prepPrepareExamples || false,  // Prep Prepare Examples
      formattedData.prepIdentifyResources || false, // Prep Identify
Resources
      formattedData.prepDevelopApproach || false,  // Prep Develop Approach
      formattedData.preparationReflection || '',   // Preparation Reflection
      formattedData.initialConnection || '',       // Initial Connection
      formattedData.concernExploration || '',      // Concern Exploration
      formattedData.developmentPlanning || '',     // Development Planning
      formattedData.docSummarizePoints || false,   // Doc Summarize Points
      formattedData.docAgreedActions || false,     // Doc Agreed Actions
      formattedData.docClarifyExpectations || false, // Doc Clarify
Expectations
      formattedData.docEstablishTimeline || false, // Doc Establish Timeline
      formattedData.formalIntervention || '',      // Formal Intervention
      formattedData.actionItems || '',             // Action Items
      formattedData.confidentialNotes || '',       // Confidential Notes
      currentUser,                                 // Created By
      new Date()                                   // Created Date
    ];

    // Add to sheet
    sheet.appendRow(rowData);

    // Format date cells in the newly added row
    const lastRow = sheet.getLastRow();
    sheet.getRange(lastRow, 2).setNumberFormat('MM/dd/yyyy');
    sheet.getRange(lastRow, 9).setNumberFormat('MM/dd/yyyy');
```

```
    sheet.getRange(lastRow, 31).setNumberFormat('MM/dd/yyyy HH:mm:ss');

    // Log the activity
    logSystemActivity('Coaching', `Added ${noteData.type} coaching note for
${noteData.counselorName}`);

    return {
      success: true,
      message: 'Coaching note saved successfully',
      noteId: noteId
    };
  } catch (error) {
    logError('CoachingService.saveCoachingNote', error);
    return {
      success: false,
      message: 'Error saving coaching note: ' + error.message
    };
  }
},
 /**
 * Send a follow-up email to a counselor after a coaching session
 * @param {Object} noteData - Data for the coaching note
 * @return {Object} Result of the operation
 */
sendCoachingFollowUpEmail: function(noteData) {
  try {
    // Validate required fields
    if (!noteData.counselorEmail || !noteData.counselorName) {
      return {
        success: false,
        message: 'Counselor email and name are required'
      };
    }
```

```
    // Get user's name for the email
    const userInfo = TeamService.getUserInfo();
    const coachName = userInfo.name || userInfo.email;

    // Set follow-up info
    let followUpInfo = 'Not yet scheduled';
    if (noteData.followUpDate) {
      const followUpDate = new Date(noteData.followUpDate);
      followUpInfo = followUpDate.toLocaleDateString();
    }

    // Create email subject and body
    const isPriority = noteData.type === 'Priority';
    const subject = `${isPriority ? 'Priority ' : ''}Coaching Session Notes -
${new Date(noteData.date).toLocaleDateString()}`;

    const htmlBody = `
      <div style="font-family: Arial, sans-serif; max-width: 600px; margin: 0
auto;">
        <h2 style="color: #001A4E;">${isPriority ? 'Priority ' : ''}Coaching
Session Summary</h2>
        <p>Hello ${noteData.counselorName},</p>
        <p>Thank you for our coaching session on ${new
Date(noteData.date).toLocaleDateString()}. Below is a summary of our
discussion and next steps.</p>

        <div style="margin: 20px 0; padding: 15px; background-color:
${isPriority ? '#F54AC' : '#FFF2DF'}; border-left: 4px solid ${isPriority ?
'#9A3499' : '#FF786E'}; border-radius: 4px; color: ${isPriority ? 'white' :
'black'};">
          <h3 style="margin-top: 0; color: ${isPriority ? 'white' :
'#001A4E'};">Coaching Focus</h3>
```

```
          <p>${noteData.focus || 'General coaching and skill
development'}</p>
        </div>

        <div style="margin: 20px 0;">
          <h3 style="color: #001A4E;">Key Discussion Points</h3>
          <p>${noteData.summary || 'We discussed your performance and
development opportunities.'}</p>
          ${noteData.concernAreas ? `<p><strong>Areas of Focus:</strong>
${noteData.concernAreas}</p>` : ''}
          ${noteData.developmentPlanning ? `<p><strong>Development
Plan:</strong> ${noteData.developmentPlanning}</p>` : ''}
        </div>

        <div style="margin: 20px 0;">
          <h3 style="color: #001A4E;">Action Items</h3>
          <p>${noteData.actionItems || 'No specific action items were
recorded.'}</p>
        </div>

        <div style="margin: 20px 0; padding: 15px; background-color: #BAE2CE;
border-radius: 4px;">
          <h3 style="margin-top: 0; color: #001A4E;">Next Steps</h3>
          <p><strong>Follow-up Date:</strong> ${followUpInfo}</p>
          <p>Please don't hesitate to reach out if you have any questions or
need additional support before our follow-up.</p>
        </div>

        <p>Best regards,<br>${coachName}</p>
      </div>
    `;

    // Send the email
```

```
    GmailApp.sendEmail(
      noteData.counselorEmail,
      subject,
      `Summary of our coaching session on ${new
Date(noteData.date).toLocaleDateString()}. Follow-up: ${followUpInfo}`,
      {
        htmlBody: htmlBody,
        name: `${coachName} (Crisis Support System)`
      }
    );

    // Log the activity
    logSystemActivity('Coaching', `Sent follow-up email to
${noteData.counselorName}`);

    return {
      success: true,
      message: 'Follow-up email sent successfully'
    };
  } catch (error) {
    logError('CoachingService.sendCoachingFollowUpEmail', error);
    return {
      success: false,
      message: 'Error sending follow-up email: ' + error.message
    };
  }
},
 /**
 * Format coaching data to ensure all required fields exist
 * @param {Object} noteData - Raw coaching data from the form
 * @return {Object} Formatted coaching data
 */
formatCoachingData: function(noteData) {
```

```
    // Create a copy of the data
    const formattedData = Object.assign({}, noteData);

    // Ensure boolean fields are actual booleans
    const booleanFields = [
      'prepGatherConcerns', 'prepReviewDocumentation', 'prepCollectEvidence',
'prepIdentifyRootCauses',
      'prepVerifyDocumentation', 'prepPrepareExamples',
'prepIdentifyResources', 'prepDevelopApproach',
      'docSummarizePoints', 'docAgreedActions', 'docClarifyExpectations',
'docEstablishTimeline'
    ];

    booleanFields.forEach(field => {
      formattedData[field] = formattedData[field] === true ||
formattedData[field] === 'true';
    });

    return formattedData;
  },
  /**
  * Convert a header name to camelCase for JavaScript object properties
  * @param {string} header - Header name from spreadsheet
  * @return {string} camelCase version of the header
  */
  camelCase: function(header) {
    // Replace special characters and split into words
    const words = header.replace(/[^\w\s]/g, ' ').split(/\s+/);

    // Convert to camelCase
    return words.map((word, index) => {
      if (index === 0) {
        return word.toLowerCase();
```

```
    }
    return word.charAt(0).toUpperCase() + word.slice(1).toLowerCase();
  }).join('');
 }
};

// Expose functions to UI
function getAllCounselors() {
 return CoachingService.getAllCounselors();
}

function getCoachingNotes(counselorEmail) {
 return CoachingService.getCoachingNotes(counselorEmail);
}

function getCoachingNoteDetails(noteId) {
 return CoachingService.getCoachingNoteDetails(noteId);
}

function saveCoachingNote(noteData) {
 return CoachingService.saveCoachingNote(noteData);
}

function sendCoachingFollowUpEmail(noteData) {
 return CoachingService.sendCoachingFollowUpEmail(noteData);
}

/**
* Shows the coaching form UI
*/
function showCoachingForm() {
 try {
   const html = HtmlService.createHtmlOutputFromFile('coaching-form')
```

```
      .setWidth(900)
      .setHeight(700)
      .setTitle('Coaching Notes');

    SpreadsheetApp.getUi().showModalDialog(html, 'Coaching Notes');
    logSystemActivity('UI', 'Opened Coaching Notes form');
  } catch (error) {
    logError('showCoachingForm', error);
    SpreadsheetApp.getUi().alert('Error opening Coaching Notes form: ' +
error.message);
  }
}

/**
* Crisis Support System for 988 Lifeline LGBTQIA+ Services
* Main entry point script for the system
*/

// Global configuration
const CRISIS_SUPPORT_CONFIG = {
  VERSION: '1.0.0',
  SHEETS: {
    COUNSELOR_TRACKING: 'Counselor Tracking',
    CALL_METRICS: 'Call Metrics',
    ALERTS: 'Alerts',
    TASKS: 'Tasks',
    SCHEDULE: 'Schedule',
    ERROR_LOG: 'Error Log',
    SYSTEM_LOG: 'System Log'
  },
  ASANA: {
    API_KEY_PROPERTY: 'asanaApiKey',
    WORKSPACE_GID_PROPERTY: 'asanaWorkspaceGid',
```

```
  PROJECT_GID_PROPERTY: 'asanaProjectGid'
 }
};

/**
* Creates the add-on menu when the spreadsheet opens
*/
function onOpen() {
 try {
   createMainMenu();
   logSystemActivity('System', 'Spreadsheet opened');
 } catch (error) {
   logError('onOpen', error);
 }
}

/**
* Creates the main menu for the Crisis Support System
*/
function createMainMenu() {
 const ui = SpreadsheetApp.getUi();
 const menu = ui.createMenu('🚨 Crisis Support');
  // Dashboard
 menu.addItem('📊 Dashboard', 'showDashboard');
 menu.addItem('🗂️ Sidebar', 'showSidebar');
  // Team Management
 menu.addSubMenu(ui.createMenu('👥 Team Management')
   .addItem('Add Counselor', 'showAddCounselorForm')
   .addItem('Update Status', 'showUpdateStatusForm')
   .addItem('Add Coaching Note', 'showCoachingForm')
   .addItem('Counselor 1:1 Notes', 'showOneOnOneNotes'));
  // Call Metrics
 menu.addSubMenu(ui.createMenu('📞 Call Metrics')
```

```
    .addItem('Enter Daily Metrics', 'showMetricsForm')
    .addItem('View Reports', 'showMetricsReport'));
  // Quality Monitoring - NEW
 menu.addSubMenu(ui.createMenu('🔍 Quality Monitoring')
    .addItem('New Quality Review', 'showQualityReviewForm')
    .addItem('View Quality Reports', 'showQualityReports'));
  // Alerts
 menu.addSubMenu(ui.createMenu('🚨 Alerts')
    .addItem('Create Alert', 'showAlertForm')
    .addItem('View Active Alerts', 'showActiveAlerts'));
  // Tasks
 menu.addSubMenu(ui.createMenu('✅ Tasks')
    .addItem('Create Task', 'showTaskForm')
    .addItem('Create Asana Task', 'showAsanaTaskForm'));
  // Schedule
 menu.addSubMenu(ui.createMenu('📅 Schedule')
    .addItem('Manage Schedule', 'showScheduleManager')
    .addItem('Initialize Week', 'showInitializeWeekForm'));
  // Team Lead Tools
 menu.addSubMenu(ui.createMenu('⏱️ Team Lead Tools')
    .addItem('Initialize Shift', 'showShiftInitialization')
    .addItem('Time Tracker', 'showTimeTracker'));
  // Support Resources
 menu.addSubMenu(ui.createMenu('🆘 Support Resources')
    .addItem('Crisis Protocols', 'showCrisisProtocols')
    .addItem('Help Resources', 'showHelp')
    .addItem('Manager 1:1 Documentation', 'showManagerOneOnOnes'));
  // Settings
 menu.addItem('⚙️ Settings', 'showSettings');
  // Add validation menu items
 addValidationMenuItems(menu);
  menu.addToUi();
}
```

```
/**
* System initialization function
*/
function initializeSystem() {
 try {
   // Check if system is already initialized
   const props = PropertiesService.getScriptProperties();
   const isInitialized = props.getProperty('systemInitialized');

   if (isInitialized === 'true') {
     const ui = SpreadsheetApp.getUi();
     const response = ui.alert(
       'System Already Initialized',
       'The Crisis Support System has already been initialized. Do you want to
re-initialize it? This may overwrite existing data.',
       ui.ButtonSet.YES_NO
     );

     if (response === ui.Button.NO) {
       return;
     }
   }

   // Create required sheets
   createRequiredSheets();

   // Set system properties
   props.setProperty('systemInitialized', 'true');
   props.setProperty('initializationDate', new Date().toISOString());

   // Show success message
   const ui = SpreadsheetApp.getUi();
```

```
    ui.alert(
      'System Initialized',
      'The Crisis Support System has been successfully initialized. You can now begin using the system.',
      ui.ButtonSet.OK
    );

    logSystemActivity('System', 'System initialized');
  } catch (error) {
    logError('initializeSystem', error);

    // Show error message
    const ui = SpreadsheetApp.getUi();
    ui.alert(
      'Initialization Error',
      'An error occurred during system initialization: ' + error.message,
      ui.ButtonSet.OK
    );
  }
}

/**
* Creates all required sheets for the system
*/
function createRequiredSheets() {
  const ss = SpreadsheetApp.getActiveSpreadsheet();
  const existingSheets = ss.getSheets().map(sheet => sheet.getName());
  const requiredSheets = ValidationSystem.getRequiredSheets();
   requiredSheets.forEach(sheetConfig => {
    if (!existingSheets.includes(sheetConfig.name)) {
      // Create sheet
      const newSheet = ss.insertSheet(sheetConfig.name);
```

```
    // Add headers if specified
    if (sheetConfig.headers && sheetConfig.headers.length > 0) {
      newSheet.appendRow(sheetConfig.headers);
      // Format header row
      newSheet.getRange(1, 1, 1, sheetConfig.headers.length)
        .setFontWeight('bold')
        .setBackground('#E0E0E0');
    }

    logSystemActivity('System', `Created sheet "${sheetConfig.name}"`);
  }
 });
}

/**
* Logs errors to the Error Log sheet
*/
function logError(functionName, error) {
 try {
  const ss = SpreadsheetApp.getActiveSpreadsheet();
  let sheet = ss.getSheetByName(CRISIS_SUPPORT_CONFIG.SHEETS.ERROR_LOG);

  // Create sheet if it doesn't exist
  if (!sheet) {
    sheet = ss.insertSheet(CRISIS_SUPPORT_CONFIG.SHEETS.ERROR_LOG);
    sheet.appendRow(['Timestamp', 'Function', 'Error Type', 'Error Message',
'Stack Trace']);
    sheet.getRange(1, 1, 1, 5)
      .setFontWeight('bold')
      .setBackground('#E0E0E0');
  }

  // Log error
```

```
    sheet.appendRow([
      new Date(),
      functionName,
      error.name || 'Error',
      error.message || 'Unknown error',
      error.stack || ''
    ]);

    // Format new row
    const lastRow = sheet.getLastRow();
    sheet.getRange(lastRow, 1, 1, 5).setWrap(true);
    sheet.getRange(lastRow,
5).setWrapStrategy(SpreadsheetApp.WrapStrategy.WRAP);

    console.error(`Error in ${functionName}: ${error.message}`);
  } catch (e) {
    // If we can't log to sheet, at least log to console
    console.error(`Failed to log error: ${e.message}`);
    console.error(`Original error in ${functionName}: ${error.message}`);
  }
}

/**
* Logs system activity to the System Log sheet
*/
function logSystemActivity(action, details) {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    let sheet = ss.getSheetByName(CRISIS_SUPPORT_CONFIG.SHEETS.SYSTEM_LOG);

    // Create sheet if it doesn't exist
    if (!sheet) {
      sheet = ss.insertSheet(CRISIS_SUPPORT_CONFIG.SHEETS.SYSTEM_LOG);
```

```
      sheet.appendRow(['Timestamp', 'Action', 'Details', 'User']);
      sheet.getRange(1, 1, 1, 4)
        .setFontWeight('bold')
        .setBackground('#E0E0E0');
    }

    // Log activity
    sheet.appendRow([
      new Date(),
      action,
      details,
      Session.getActiveUser().getEmail()
    ]);
  } catch (e) {
    console.error(`Failed to log system activity: ${e.message}`);
  }
}

/**
* Service for handling data retrieval and manipulation
*/
const DataService = {
  /**
   * Gets metrics for the dashboard
   * @return {object} Dashboard metrics
   */
  getDashboardMetrics: function() {
    try {
      const ss = SpreadsheetApp.getActiveSpreadsheet();
      const metricsSheet =
ss.getSheetByName(CRISIS_SUPPORT_CONFIG.SHEETS.CALL_METRICS);

      // Return placeholder metrics if sheet doesn't exist
```

```
    if (!metricsSheet) {
      return {
        answerRate: '95.0%',
        qualityScore: '72.5',
        avgConversations: '8.3',
        utoRate: '3.7%'
      };
    }

    // Get data from the last 30 days
    const today = new Date();
    const thirtyDaysAgo = new Date();
    thirtyDaysAgo.setDate(today.getDate() - 30);

    const data = metricsSheet.getDataRange().getValues();

    // Skip if only header row
    if (data.length <= 1) {
      return {
        answerRate: '--',
        qualityScore: '--',
        avgConversations: '--',
        utoRate: '--'
      };
    }

    // Find column indices
    const headers = data[0];
    const dateIndex = headers.indexOf('Date');
    const callsOfferedIndex = headers.indexOf('Calls Offered');
    const callsAcceptedIndex = headers.indexOf('Calls Accepted');
    const talkTimeIndex = headers.indexOf('Talk Time (min)');
    const afterCallWorkIndex = headers.indexOf('After Call Work (min)');
```

```
    const onQueueIndex = headers.indexOf('On Queue %');

    // Filter for last 30 days' data
    const recentData = data.slice(1).filter(row => {
      const rowDate = new Date(row[dateIndex]);
      return rowDate >= thirtyDaysAgo && rowDate <= today;
    });

    // Calculate metrics
    let totalCallsOffered = 0;
    let totalCallsAccepted = 0;
    let totalTalkTime = 0;
    let totalRows = recentData.length;
    let totalOnQueue = 0;

    recentData.forEach(row => {
      totalCallsOffered += row[callsOfferedIndex] || 0;
      totalCallsAccepted += row[callsAcceptedIndex] || 0;
      totalTalkTime += row[talkTimeIndex] || 0;
      totalOnQueue += row[onQueueIndex] || 0;
    });

    // Format metrics
    const answerRate = totalCallsOffered > 0 ?
      ((totalCallsAccepted / totalCallsOffered) * 100).toFixed(1) + '%' :
'--';

    const avgConversations = totalRows > 0 ?
      (totalCallsAccepted / totalRows).toFixed(1) : '--';

    // Get quality score from somewhere else
    const qualityScore = '72.5'; // Placeholder until implemented
```

```
    // Get UTO rate from somewhere else
    const utoRate = '3.7%'; // Placeholder until implemented

    return {
      answerRate: answerRate,
      qualityScore: qualityScore,
      avgConversations: avgConversations,
      utoRate: utoRate,
      // Additional metrics as needed
      callsOffered: totalCallsOffered.toString(),
      callsAccepted: totalCallsAccepted.toString(),
      avgTalkTime: totalRows > 0 ? (totalTalkTime /
totalCallsAccepted).toFixed(1) : '--',
      onQueuePercent: totalRows > 0 ? (totalOnQueue / totalRows).toFixed(1) +
'%' : '--'
    };
  } catch (error) {
    logError('getDashboardMetrics', error);
    return {
      answerRate: '--',
      qualityScore: '--',
      avgConversations: '--',
      utoRate: '--'
    };
  }
},
 /**
 * Gets active alerts for the dashboard
 * @param {number} limit - Maximum number of alerts to return
 * @return {Array} Active alerts
 */
getActiveAlerts: function(limit = 5) {
  try {
```

```
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    const sheet = ss.getSheetByName(CRISIS_SUPPORT_CONFIG.SHEETS.ALERTS);

    // Return empty array if sheet doesn't exist
    if (!sheet) {
      return [];
    }

    // Get all alerts
    const data = sheet.getDataRange().getValues();

    // Skip if only header row
    if (data.length <= 1) {
      return [];
    }

    // Find column indices
    const headers = data[0];
    const dateIndex = headers.indexOf('Date');
    const severityIndex = headers.indexOf('Severity');
    const messageIndex = headers.indexOf('Message');
    const categoryIndex = headers.indexOf('Category');
    const statusIndex = headers.indexOf('Status');

    // Filter active alerts
    const activeAlerts = data.slice(1)
      .filter(row => row[statusIndex] !== 'Resolved' && row[statusIndex] !==
'Closed')
      .map(row => ({
        date: row[dateIndex],
        severity: row[severityIndex],
        message: row[messageIndex],
        category: row[categoryIndex],
```

```
        status: row[statusIndex]
      }))
      .sort((a, b) => {
        // Sort by severity (High > Medium > Low)
        const severityOrder = { 'High': 0, 'Medium': 1, 'Low': 2 };
        const severityA = severityOrder[a.severity] || 3;
        const severityB = severityOrder[b.severity] || 3;

        // Then by date (newest first)
        if (severityA === severityB) {
          return new Date(b.date) - new Date(a.date);
        }

        return severityA - severityB;
      })
      .slice(0, limit);

    return activeAlerts;
  } catch (error) {
    logError('getActiveAlerts', error);
    return [];
  }
},
 /**
 * Gets pending tasks for the dashboard
 * @param {number} limit - Maximum number of tasks to return
 * @return {Array} Pending tasks
 */
getPendingTasks: function(limit = 5) {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    const sheet = ss.getSheetByName(CRISIS_SUPPORT_CONFIG.SHEETS.TASKS);
```

```
    // Return empty array if sheet doesn't exist
    if (!sheet) {
      return [];
    }

    // Get all tasks
    const data = sheet.getDataRange().getValues();

    // Skip if only header row
    if (data.length <= 1) {
      return [];
    }

    // Find column indices
    const headers = data[0];
    const taskIndex = headers.indexOf('Task');
    const descriptionIndex = headers.indexOf('Description');
    const assigneeIndex = headers.indexOf('Assignee');
    const dueDateIndex = headers.indexOf('Due Date');
    const priorityIndex = headers.indexOf('Priority');
    const statusIndex = headers.indexOf('Status');

    // Filter pending tasks
    const pendingTasks = data.slice(1)
      .filter(row => row[statusIndex] !== 'Completed' && row[statusIndex] !==
'Closed')
      .map(row => ({
        task: row[taskIndex],
        description: row[descriptionIndex],
        assignee: row[assigneeIndex],
        dueDate: row[dueDateIndex],
        priority: row[priorityIndex],
        status: row[statusIndex]
```

```
      }))
      .sort((a, b) => {
        // Sort by priority (High > Medium > Low)
        const priorityOrder = { 'High': 0, 'Medium': 1, 'Low': 2 };
        const priorityA = priorityOrder[a.priority] || 3;
        const priorityB = priorityOrder[b.priority] || 3;

        // Then by due date (soonest first)
        if (priorityA === priorityB) {
          return new Date(a.dueDate) - new Date(b.dueDate);
        }

        return priorityA - priorityB;
      })
      .slice(0, limit);

    return pendingTasks;
  } catch (error) {
    logError('getPendingTasks', error);
    return [];
  }
},
 /**
 * Gets team status for the dashboard
 * @return {Array} Counselor status data
 */
getTeamStatus: function() {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    const sheet =
ss.getSheetByName(CRISIS_SUPPORT_CONFIG.SHEETS.COUNSELOR_TRACKING);

    // Return empty array if sheet doesn't exist
```

```
    if (!sheet) {
      return [];
    }

    // Get all counselor data
    const data = sheet.getDataRange().getValues();

    // Skip if only header row
    if (data.length <= 1) {
      return [];
    }

    // Find column indices
    const headers = data[0];
    const nameIndex = headers.indexOf('Name');
    const emailIndex = headers.indexOf('Email');
    const statusIndex = headers.indexOf('Status');
    const teamIndex = headers.indexOf('Team');

    // Get counselor status data
    const counselors = data.slice(1)
      .map(row => ({
        name: row[nameIndex],
        email: row[emailIndex],
        status: row[statusIndex],
        team: row[teamIndex]
      }))
      .filter(counselor => counselor.name && counselor.status) // Filter out
empty rows
      .sort((a, b) => a.name.localeCompare(b.name)); // Sort by name

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

```
    logError('getTeamStatus', error);
    return [];
  }
},
 /**
 * Gets performance data for charts
 * @param {string} timeframe - Week or month
 * @return {object} Performance data for charts
 */
getPerformanceData: function(timeframe = 'week') {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    const metricsSheet =
ss.getSheetByName(CRISIS_SUPPORT_CONFIG.SHEETS.CALL_METRICS);

    // Return null if sheet doesn't exist
    if (!metricsSheet) {
      return null;
    }

    // Set date range based on timeframe
    const endDate = new Date();
    const startDate = new Date();

    if (timeframe === 'week') {
      startDate.setDate(endDate.getDate() - 7);
    } else if (timeframe === 'month') {
      startDate.setDate(endDate.getDate() - 30);
    } else {
      startDate.setDate(endDate.getDate() - 7); // Default to week
    }

    // Get metrics data
```

```
    const data = metricsSheet.getDataRange().getValues();

    // Skip if only header row
    if (data.length <= 1) {
      return null;
    }

    // Find column indices
    const headers = data[0];
    const dateIndex = headers.indexOf('Date');
    const callsOfferedIndex = headers.indexOf('Calls Offered');
    const callsAcceptedIndex = headers.indexOf('Calls Accepted');
    const talkTimeIndex = headers.indexOf('Talk Time (min)');
    const afterCallWorkIndex = headers.indexOf('After Call Work (min)');
    const onQueueIndex = headers.indexOf('On Queue %');

    // Filter for date range
    const filteredData = data.slice(1)
      .filter(row => {
        const rowDate = new Date(row[dateIndex]);
        return rowDate >= startDate && rowDate <= endDate;
      })
      .map(row => ({
        date: new Date(row[dateIndex]).toLocaleDateString(),
        callsOffered: row[callsOfferedIndex] || 0,
        callsAccepted: row[callsAcceptedIndex] || 0,
        answerRate: row[callsOfferedIndex] > 0 ?
          ((row[callsAcceptedIndex] / row[callsOfferedIndex]) *
100).toFixed(1) : 0,
        talkTime: row[talkTimeIndex] || 0,
        afterCallWork: row[afterCallWorkIndex] || 0,
        onQueue: row[onQueueIndex] || 0
      }))
```

```
      .sort((a, b) => new Date(a.date) - new Date(b.date));

    // Prepare chart data
    const chartData = {
      labels: filteredData.map(item => item.date),
      answerRates: filteredData.map(item => item.answerRate),
      callVolumes: filteredData.map(item => item.callsAccepted),
      onQueueRates: filteredData.map(item => item.onQueue)
    };

    return chartData;
  } catch (error) {
    logError('getPerformanceData', error);
    return null;
  }
},
 /**
 * Gets current user info
 * @return {object} User info
 */
getCurrentUserInfo: function() {
  try {
    const email = Session.getActiveUser().getEmail();
    const userName = email.split('@')[0]; // Basic name extraction from email

    // Try to find user in counselor tracking
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    const sheet =
ss.getSheetByName(CRISIS_SUPPORT_CONFIG.SHEETS.COUNSELOR_TRACKING);

    if (sheet) {
      const data = sheet.getDataRange().getValues();
      const headers = data[0];
```

```
      const nameIndex = headers.indexOf('Name');
      const emailIndex = headers.indexOf('Email');

      // Look for matching email
      for (let i = 1; i < data.length; i++) {
        if (data[i][emailIndex] === email) {
          return {
            email: email,
            name: data[i][nameIndex]
          };
        }
      }
    }

    // Return basic info if not found in tracking
    return {
      email: email,
      name: userName
    };
  } catch (error) {
    logError('getCurrentUserInfo', error);
    return {
      email: Session.getActiveUser().getEmail(),
      name: 'Team Lead'
    };
  }
},
 /**
 * Gets a list of all counselors
 * @return {Array} List of counselors
 */
getAllCounselors: function() {
  try {
```

```
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    const sheet =
ss.getSheetByName(CRISIS_SUPPORT_CONFIG.SHEETS.COUNSELOR_TRACKING);

    // Return empty array if sheet doesn't exist
    if (!sheet) {
      return [];
    }

    // Get all counselor data
    const data = sheet.getDataRange().getValues();

    // Skip if only header row
    if (data.length <= 1) {
      return [];
    }

    // Find column indices
    const headers = data[0];
    const nameIndex = headers.indexOf('Name');
    const emailIndex = headers.indexOf('Email');
    const statusIndex = headers.indexOf('Status');
    const teamIndex = headers.indexOf('Team');

    // Get counselor data
    const counselors = data.slice(1)
      .map(row => ({
        name: row[nameIndex],
        email: row[emailIndex],
        status: row[statusIndex],
        team: row[teamIndex]
      }))
      .filter(counselor => counselor.name) // Filter out empty rows
```

```
      .sort((a, b) => a.name.localeCompare(b.name)); // Sort by name

    return counselors;
  } catch (error) {
    logError('getAllCounselors', error);
    return [];
  }
},
 /**
 * Creates a new counselor entry
 * @param {object} counselorData - Data for the new counselor
 * @return {object} Result of the operation
 */
createCounselor: function(counselorData) {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    let sheet =
ss.getSheetByName(CRISIS_SUPPORT_CONFIG.SHEETS.COUNSELOR_TRACKING);

    // Create sheet if it doesn't exist
    if (!sheet) {
      sheet =
ss.insertSheet(CRISIS_SUPPORT_CONFIG.SHEETS.COUNSELOR_TRACKING);
      sheet.appendRow([
        'Date Added',
        'Name',
        'Email',
        'Phone',
        'Status',
        'Team',
        'Start Date',
        'Goal/Focus',
        'Notes'
```

```
    ]);
    sheet.getRange(1, 1, 1, 9)
      .setFontWeight('bold')
      .setBackground('#E0E0E0');
  }

  // Check if counselor already exists
  const data = sheet.getDataRange().getValues();
  const headers = data[0];
  const emailIndex = headers.indexOf('Email');

  for (let i = 1; i < data.length; i++) {
    if (data[i][emailIndex] === counselorData.email) {
      return {
        success: false,
        message: 'A counselor with this email already exists'
      };
    }
  }

  // Add new counselor
  sheet.appendRow([
    new Date(), // Date Added
    counselorData.name,
    counselorData.email,
    counselorData.phone || '',
    counselorData.status || 'Active',
    counselorData.team || '',
    counselorData.startDate || '',
    counselorData.goalFocus || '',
    counselorData.notes || ''
  ]);
```

```
    // Log the activity
    logSystemActivity('Counselor Management', `Added new counselor:
${counselorData.name}`);

    return {
      success: true,
      message: 'Counselor added successfully'
    };
  } catch (error) {
    logError('createCounselor', error);
    return {
      success: false,
      message: 'Error adding counselor: ' + error.message
    };
  }
},
 /**
 * Updates counselor status
 * @param {object} updateData - Data for the status update
 * @return {object} Result of the operation
 */
updateCounselorStatus: function(updateData) {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    const sheet =
ss.getSheetByName(CRISIS_SUPPORT_CONFIG.SHEETS.COUNSELOR_TRACKING);

    // Return error if sheet doesn't exist
    if (!sheet) {
      return {
        success: false,
        message: 'Counselor tracking sheet not found'
      };
```

```
    }

    // Find counselor by email
    const data = sheet.getDataRange().getValues();
    const headers = data[0];
    const nameIndex = headers.indexOf('Name');
    const emailIndex = headers.indexOf('Email');
    const statusIndex = headers.indexOf('Status');

    let rowIndex = -1;

    for (let i = 1; i < data.length; i++) {
      if (data[i][emailIndex] === updateData.email) {
        rowIndex = i + 1; // +1 because array is 0-indexed but sheet is
1-indexed
        break;
      }
    }

    // Return error if counselor not found
    if (rowIndex === -1) {
      return {
        success: false,
        message: 'Counselor not found'
      };
    }

    // Update status
    sheet.getRange(rowIndex, statusIndex + 1).setValue(updateData.status);

    // Log the activity
    logSystemActivity('Counselor Management', `Updated status for
${data[rowIndex-1][nameIndex]} to ${updateData.status}`);
```

```
      return {
        success: true,
        message: 'Counselor status updated successfully'
      };
    } catch (error) {
      logError('updateCounselorStatus', error);
      return {
        success: false,
        message: 'Error updating counselor status: ' + error.message
      };
    }
  }
};

// Expose functions to UI
function getDashboardMetrics() {
  return DataService.getDashboardMetrics();
}

function getActiveAlerts(limit) {
  return DataService.getActiveAlerts(limit);
}

function getPendingTasks(limit) {
  return DataService.getPendingTasks(limit);
}

function getTeamStatus() {
  return DataService.getTeamStatus();
}

function getPerformanceData(timeframe) {
```

```
 return DataService.getPerformanceData(timeframe);
}

function getCurrentUserInfo() {
 return DataService.getCurrentUserInfo();
}

function getAllCounselors() {
 return DataService.getAllCounselors();
}

function createCounselor(counselorData) {
 return DataService.createCounselor(counselorData);
}

function updateCounselorStatus(updateData) {
 return DataService.updateCounselorStatus(updateData);
}
/**
* ManagerOneOnOneService.gs
* Service for managing 1:1 meetings with managers in the Crisis Support System
for 988 Lifeline LGBTQIA+ Services
* Handles creation, retrieval, and management of manager 1:1 meetings and
action items
*/

const ManagerOneOnOneService = {
 /**
  * Saves a new manager 1:1 meeting
  * @param {Object} meetingData - The meeting data object
  * @return {Object} Result object with success status and message
  */
 saveManagerOneOnOne: function(meetingData) {
```

```
  try {
    // Validate required fields
    if (!meetingData.meetingDate || !meetingData.managerName) {
      return {
        success: false,
        message: "Required fields missing. Please provide meeting date and
manager name."
      };
    }

    const ss = SpreadsheetApp.getActiveSpreadsheet();
    let sheet = ss.getSheetByName('Manager 1:1 Notes');

    // Create sheet if it doesn't exist
    if (!sheet) {
      sheet = ss.insertSheet('Manager 1:1 Notes');

      // Add headers
      sheet.appendRow([
        'ID',
        'Meeting Date',
        'Next Check-In Date',
        'Manager Name',
        'Feeling Overall',
        'Work-Life Balance',
        'Accomplishments',
        'Feedback Reflection',
        'Recent Challenges',
        'Blockers',
        'Growth Areas',
        'Support Needed',
        'Action Items',
        'Created By',
```

```
        'Created Date',
        'Last Modified',
        'Last Modified By'
      ]);

      // Format header row
      sheet.getRange(1, 1, 1, 17)
        .setFontWeight('bold')
        .setBackground('#E0E0E0');
    }

    // Generate a unique ID for the meeting
    const meetingId = Utilities.getUuid();

    // Convert action items to JSON string for storage
    const actionItemsJson = JSON.stringify(meetingData.actionItems || []);

    // Add meeting to sheet
    sheet.appendRow([
      meetingId,
      new Date(meetingData.meetingDate),
      meetingData.nextCheckInDate ? new Date(meetingData.nextCheckInDate) :
'',
      meetingData.managerName,
      meetingData.feelingOverall || '',
      meetingData.workLifeBalance || '',
      meetingData.accomplishments || '',
      meetingData.feedbackReflection || '',
      meetingData.recentChallenges || '',
      meetingData.blockers || '',
      meetingData.growthAreas || '',
      meetingData.supportNeeded || '',
      actionItemsJson,
```

```
      Session.getActiveUser().getEmail(),
      new Date(),
      new Date(),
      Session.getActiveUser().getEmail()
    ]);

    // Format date cells in the newly added row
    const lastRow = sheet.getLastRow();
    sheet.getRange(lastRow, 2, 1, 2).setNumberFormat("MM/dd/yyyy");
    sheet.getRange(lastRow, 15, 1, 2).setNumberFormat("MM/dd/yyyy HH:mm:ss");

    // Create action items in separate sheet if there are any
    if (meetingData.actionItems && meetingData.actionItems.length > 0) {
      this.createActionItems(meetingId, new Date(meetingData.meetingDate),
meetingData.actionItems);
    }

    // Log the activity
    logSystemActivity("Manager 1:1", "Created 1:1 meeting with " +
meetingData.managerName);

    return {
      success: true,
      message: "Meeting saved successfully"
    };
  } catch (error) {
    logError("ManagerOneOnOneService.saveManagerOneOnOne", error);
    return {
      success: false,
      message: "Error saving meeting: " + error.message
    };
  }
},
```

```
 /**
 * Creates action items in the Action Items sheet
 * @param {string} meetingId - The ID of the associated meeting
 * @param {Date} meetingDate - The date of the meeting
 * @param {Array} actionItems - Array of action item objects
 */
createActionItems: function(meetingId, meetingDate, actionItems) {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    let sheet = ss.getSheetByName('Manager 1:1 Action Items');

    // Create sheet if it doesn't exist
    if (!sheet) {
      sheet = ss.insertSheet('Manager 1:1 Action Items');

      // Add headers
      sheet.appendRow([
        'ID',
        'Meeting ID',
        'Meeting Date',
        'Description',
        'Due Date',
        'Status',
        'Completed Date',
        'Completed By',
        'Created By',
        'Created Date'
      ]);

      // Format header row
      sheet.getRange(1, 1, 1, 10)
        .setFontWeight('bold')
        .setBackground('#E0E0E0');
```

```
    }

    // Add each action item to the sheet
    actionItems.forEach(item => {
      const actionItemId = Utilities.getUuid();

      sheet.appendRow([
        actionItemId,
        meetingId,
        meetingDate,
        item.description,
        item.dueDate ? new Date(item.dueDate) : '',
        item.status || 'pending',
        '',
        '',
        Session.getActiveUser().getEmail(),
        new Date()
      ]);

      // Format date cells in the newly added row
      const lastRow = sheet.getLastRow();
      sheet.getRange(lastRow, 3).setNumberFormat("MM/dd/yyyy");
      sheet.getRange(lastRow, 5).setNumberFormat("MM/dd/yyyy");
      sheet.getRange(lastRow, 7).setNumberFormat("MM/dd/yyyy HH:mm:ss");
      sheet.getRange(lastRow, 10).setNumberFormat("MM/dd/yyyy HH:mm:ss");
    });
  } catch (error) {
    logError("ManagerOneOnOneService.createActionItems", error);
    throw error;
  }
},
 /**
 * Gets recent and upcoming manager 1:1 meetings
```

```
  * @param {number} daysBack - Number of days to look back
  * @param {number} daysForward - Number of days to look forward
  * @return {Array} Array of meeting objects
  */
 getRecentAndUpcomingManagerOneOnOnes: function(daysBack = 30, daysForward =
30) {
   try {
     const ss = SpreadsheetApp.getActiveSpreadsheet();
     const sheet = ss.getSheetByName('Manager 1:1 Notes');

     if (!sheet) {
       return [];
     }

     // Calculate date range
     const today = new Date();
     const pastDate = new Date();
     pastDate.setDate(today.getDate() - daysBack);
     const futureDate = new Date();
     futureDate.setDate(today.getDate() + daysForward);

     // Get all meetings
     const dataRange = sheet.getDataRange();
     const values = dataRange.getValues();

     // Skip the header row
     const meetings = [];
     for (let i = 1; i < values.length; i++) {
       const meetingDate = values[i][1]; // Meeting Date column
       // Only include meetings within the date range
       if (meetingDate >= pastDate && meetingDate <= futureDate) {
         meetings.push(this.formatMeetingData(values[i]));
       }
```

```
    }

    // Sort by meeting date (newest first)
    return meetings.sort((a, b) => new Date(b.meetingDate) - new
Date(a.meetingDate));
  } catch (error) {
    logError("ManagerOneOnOneService.getRecentAndUpcomingManagerOneOnOnes",
error);
    return [];
  }
},
 /**
 * Gets all manager 1:1 meetings (for history view)
 * @return {Array} Array of meeting objects
 */
getManagerOneOnOneHistory: function() {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    const sheet = ss.getSheetByName('Manager 1:1 Notes');

    if (!sheet) {
      return [];
    }

    // Get all meetings
    const dataRange = sheet.getDataRange();
    const values = dataRange.getValues();

    // Skip the header row
    const meetings = [];
    for (let i = 1; i < values.length; i++) {
      meetings.push(this.formatMeetingData(values[i]));
    }
```

```
    // Sort by meeting date (newest first)
    return meetings.sort((a, b) => new Date(b.meetingDate) - new
Date(a.meetingDate));
  } catch (error) {
    logError("ManagerOneOnOneService.getManagerOneOnOneHistory", error);
    return [];
  }
},
 /**
 * Gets action items from manager 1:1 meetings
 * @param {string} status - Filter by status (optional)
 * @return {Array} Array of action item objects
 */
getManagerOneOnOneActionItems: function(status = null) {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    const sheet = ss.getSheetByName('Manager 1:1 Action Items');

    if (!sheet) {
      return [];
    }

    // Get all action items
    const dataRange = sheet.getDataRange();
    const values = dataRange.getValues();

    // Skip the header row
    const actionItems = [];
    for (let i = 1; i < values.length; i++) {
      const item = {
        id: values[i][0],
        meetingId: values[i][1],
```

```
        meetingDate: values[i][2],
        description: values[i][3],
        dueDate: values[i][4],
        status: values[i][5],
        completedDate: values[i][6],
        completedBy: values[i][7],
        createdBy: values[i][8],
        createdDate: values[i][9]
      };

      // Check if this item is overdue
      if (item.status !== 'completed' && item.dueDate && item.dueDate < new
Date()) {
        item.status = 'overdue';
      }

      // Filter by status if provided
      if (!status || item.status === status) {
        actionItems.push(item);
      }
    }

    // Sort by due date (nearest first, then overdue items at the top)
    return actionItems.sort((a, b) => {
      // Overdue items first
      if (a.status === 'overdue' && b.status !== 'overdue') return -1;
      if (a.status !== 'overdue' && b.status === 'overdue') return 1;

      // Then by due date
      if (a.dueDate && b.dueDate) return new Date(a.dueDate) - new
Date(b.dueDate);
      if (a.dueDate && !b.dueDate) return -1;
      if (!a.dueDate && b.dueDate) return 1;
```

```
      // Then by meeting date
      return new Date(b.meetingDate) - new Date(a.meetingDate);
    });
  } catch (error) {
    logError("ManagerOneOnOneService.getManagerOneOnOneActionItems", error);
    return [];
  }
},
 /**
 * Updates an action item status
 * @param {string} actionItemId - The ID of the action item
 * @param {string} newStatus - The new status
 * @return {Object} Result object with success status and message
 */
updateManagerOneOnOneActionItemStatus: function(actionItemId, newStatus) {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    const sheet = ss.getSheetByName('Manager 1:1 Action Items');

    if (!sheet) {
      return {
        success: false,
        message: "Action Items sheet not found"
      };
    }

    // Find the action item by ID
    const dataRange = sheet.getDataRange();
    const values = dataRange.getValues();
    let rowIndex = -1;

    for (let i = 0; i < values.length; i++) {
```

```
      if (values[i][0] === actionItemId) {
        rowIndex = i + 1; // +1 because array is 0-indexed but sheet is
1-indexed
        break;
      }
    }

    if (rowIndex === -1) {
      return {
        success: false,
        message: "Action Item not found"
      };
    }

    // Update the status
    sheet.getRange(rowIndex, 6).setValue(newStatus);

    // If completed, update completed date and user
    if (newStatus === 'completed') {
      sheet.getRange(rowIndex, 7).setValue(new Date());
      sheet.getRange(rowIndex,
8).setValue(Session.getActiveUser().getEmail());
    } else {
      // If reopened, clear completed date and user
      sheet.getRange(rowIndex, 7).setValue('');
      sheet.getRange(rowIndex, 8).setValue('');
    }

    // Log the activity
    const actionDescription = values[rowIndex - 1][3];
    logSystemActivity("Manager 1:1", `Updated action item status:
"${actionDescription}" to ${newStatus}`);
```

```
      return {
        success: true,
        message: "Action item status updated successfully"
      };
    } catch (error) {
      logError("ManagerOneOnOneService.updateManagerOneOnOneActionItemStatus",
error);
      return {
        success: false,
        message: "Error updating action item status: " + error.message
      };
    }
  },
  /**
  * Helper function to format meeting data from sheet values
  * @param {Array} row - Row values from sheet
  * @return {Object} Formatted meeting object
  */
  formatMeetingData: function(row) {
    // Parse action items from JSON string
    let actionItems = [];
    try {
      if (row[12]) {
        actionItems = JSON.parse(row[12]);
      }
    } catch (e) {
      console.error('Error parsing action items JSON:', e);
    }

    return {
      id: row[0],
      meetingDate: row[1],
      nextCheckInDate: row[2],
```

```
      managerName: row[3],
      feelingOverall: row[4],
      workLifeBalance: row[5],
      accomplishments: row[6],
      feedbackReflection: row[7],
      recentChallenges: row[8],
      blockers: row[9],
      growthAreas: row[10],
      supportNeeded: row[11],
      actionItems: actionItems,
      createdBy: row[13],
      createdDate: row[14],
      lastModified: row[15],
      lastModifiedBy: row[16]
    };
  },
  /**
  * Gets the current user's email
  * @return {string} Current user email
  */
  getCurrentUserEmail: function() {
    return Session.getActiveUser().getEmail();
  }
};

/**
* Global function to save a manager 1:1 meeting (for UI calls)
* @param {Object} meetingData - The meeting data
* @return {Object} Result object
*/
function saveManagerOneOnOne(meetingData) {
  return ManagerOneOnOneService.saveManagerOneOnOne(meetingData);
}
```

```
/**
* Global function to get recent and upcoming 1:1 meetings (for UI calls)
* @return {Array} Array of meeting objects
*/
function getRecentAndUpcomingManagerOneOnOnes() {
 return ManagerOneOnOneService.getRecentAndUpcomingManagerOneOnOnes();
}

/**
* Global function to get meeting history (for UI calls)
* @return {Array} Array of meeting objects
*/
function getManagerOneOnOneHistory() {
 return ManagerOneOnOneService.getManagerOneOnOneHistory();
}

/**
* Global function to get action items (for UI calls)
* @return {Array} Array of action item objects
*/
function getManagerOneOnOneActionItems() {
 return ManagerOneOnOneService.getManagerOneOnOneActionItems();
}

/**
* Global function to update an action item status (for UI calls)
* @param {string} actionItemId - The ID of the action item
* @param {string} newStatus - The new status
* @return {Object} Result object
*/
function updateManagerOneOnOneActionItemStatus(actionItemId, newStatus) {
```

```
 return
ManagerOneOnOneService.updateManagerOneOnOneActionItemStatus(actionItemId,
newStatus);
}

/**
* Global function to get the current user's email (for UI calls)
* @return {string} Current user email
*/
function getCurrentUserEmail() {
 return ManagerOneOnOneService.getCurrentUserEmail();
}

/**
* Shows the Manager 1:1 Documentation UI
*/
function showManagerOneOnOnes() {
 try {
   const html = HtmlService.createHtmlOutputFromFile('manager-one-on-ones')
     .setWidth(800)
     .setHeight(600)
     .setTitle('Manager 1:1 Documentation');

   SpreadsheetApp.getUi().showModalDialog(html, 'Manager 1:1 Documentation');
   logSystemActivity('UI', 'Opened Manager 1:1 Documentation');
 } catch (error) {
   logError('showManagerOneOnOnes', error);
   SpreadsheetApp.getUi().alert('Error opening Manager 1:1 Documentation: ' +
error.message);
 }
}
/**
* MetricsService.gs
```

```
*
* Service for managing call metrics functionality
*/
const MetricsService = {
 /**
  * Adds daily metrics to the system
  * @param {object} metricsData - Daily metrics data
  * @return {object} Result of the operation
  */
 addDailyMetrics: function(metricsData) {
   try {
     const ss = SpreadsheetApp.getActiveSpreadsheet();
     let sheet = ss.getSheetByName(CRISIS_SUPPORT_CONFIG.SHEETS.CALL_METRICS);

     // Create sheet if it doesn't exist
     if (!sheet) {
       sheet = ss.insertSheet(CRISIS_SUPPORT_CONFIG.SHEETS.CALL_METRICS);
       sheet.appendRow([
         'Date',
         'Calls Offered',
         'Calls Accepted',
         'Talk Time (min)',
         'After Call Work (min)',
         'On Queue %',
         'Behaviors Impacting Performance'
       ]);
       sheet.getRange(1, 1, 1, 7)
         .setFontWeight('bold')
         .setBackground('#E0E0E0');
     }

     // Validate required fields
```

```
    if (!metricsData.date || !metricsData.callsOffered ||
!metricsData.callsAccepted) {
      return {
        success: false,
        message: 'Date, calls offered, and calls accepted are required'
      };
    }

    // Prepare row data
    const rowData = [
      metricsData.date instanceof Date ? metricsData.date : new
Date(metricsData.date),
      metricsData.callsOffered,
      metricsData.callsAccepted,
      metricsData.talkTime || 0,
      metricsData.afterCallWork || 0,
      metricsData.onQueuePercent || 0,
      metricsData.behaviors || ''
    ];

    // Add to sheet
    sheet.appendRow(rowData);

    // Log activity
    logSystemActivity('Call Metrics', `Added metrics for date:
${metricsData.date}`);

    return {
      success: true,
      message: 'Metrics added successfully'
    };
  } catch (error) {
    logError('addDailyMetrics', error);
```

```
    return {
      success: false,
      message: 'Error adding metrics: ' + error.message
    };
  }
},
 /**
 * Gets metrics from the system
 * @param {object} options - Options for filtering metrics
 * @return {Array} Metrics data
 */
getMetrics: function(options = {}) {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    const sheet =
ss.getSheetByName(CRISIS_SUPPORT_CONFIG.SHEETS.CALL_METRICS);

    if (!sheet) {
      return [];
    }

    // Get all data
    const data = sheet.getDataRange().getValues();

    // Skip header row
    if (data.length <= 1) {
      return [];
    }

    // Get column indices
    const headers = data[0];
    const dateIndex = headers.indexOf('Date');
    const callsOfferedIndex = headers.indexOf('Calls Offered');
```

```
    const callsAcceptedIndex = headers.indexOf('Calls Accepted');
    const talkTimeIndex = headers.indexOf('Talk Time (min)');
    const afterCallWorkIndex = headers.indexOf('After Call Work (min)');
    const onQueuePercentIndex = headers.indexOf('On Queue %');
    const behaviorsIndex = headers.indexOf('Behaviors Impacting
Performance');

    // Check if required columns exist
    if (dateIndex === -1 || callsOfferedIndex === -1 || callsAcceptedIndex
=== -1) {
      throw new Error('Required columns not found in Call Metrics sheet');
    }

    // Map data to metrics objects
    let metrics = [];

    for (let i = 1; i < data.length; i++) {
      const row = data[i];

      // Skip empty rows
      if (!row[dateIndex]) {
        continue;
      }

      const metric = {
        date: row[dateIndex],
        callsOffered: row[callsOfferedIndex] || 0,
        callsAccepted: row[callsAcceptedIndex] || 0,
        talkTime: talkTimeIndex !== -1 ? row[talkTimeIndex] || 0 : 0,
        afterCallWork: afterCallWorkIndex !== -1 ? row[afterCallWorkIndex] ||
0 : 0,
        onQueuePercent: onQueuePercentIndex !== -1 ? row[onQueuePercentIndex]
|| 0 : 0,
```

```
        behaviors: behaviorsIndex !== -1 ? row[behaviorsIndex] || '' : ''
      };

      // Calculate answer rate
      metric.answerRate = metric.callsOffered > 0 ?
        (metric.callsAccepted / metric.callsOffered * 100) : 0;

      metrics.push(metric);
    }

    // Filter by date range if specified
    if (options.startDate && options.endDate) {
      const startDate = new Date(options.startDate);
      const endDate = new Date(options.endDate);

      metrics = metrics.filter(metric => {
        const metricDate = metric.date instanceof Date ?
          metric.date : new Date(metric.date);

        return metricDate >= startDate && metricDate <= endDate;
      });
    }

    return metrics;
  } catch (error) {
    logError('getMetrics', error);
    return [];
  }
},
 /**
 * Gets metrics report data with summary information
 * @param {object} options - Options for filtering metrics
 * @return {object} Metrics report data
```

```
 */
getMetricsReport: function(options = {}) {
  try {
    // Get raw metrics data
    const metrics = this.getMetrics(options);

    if (metrics.length === 0) {
      return {
        success: false,
        message: 'No metrics data found for the specified period'
      };
    }

    // Calculate summary metrics
    const totalCallsOffered = metrics.reduce((sum, metric) => sum +
metric.callsOffered, 0);
    const totalCallsAccepted = metrics.reduce((sum, metric) => sum +
metric.callsAccepted, 0);
    const totalTalkTime = metrics.reduce((sum, metric) => sum +
metric.talkTime, 0);
    const totalAfterCallWork = metrics.reduce((sum, metric) => sum +
metric.afterCallWork, 0);
    const totalOnQueuePercent = metrics.reduce((sum, metric) => sum +
metric.onQueuePercent, 0);

    // Create summary object
    const summary = {
      totalCalls: totalCallsOffered,
      answerRate: totalCallsOffered > 0 ? (totalCallsAccepted /
totalCallsOffered * 100) : 0,
      avgTalkTime: totalCallsAccepted > 0 ? (totalTalkTime /
totalCallsAccepted) : 0,
```

```
      afterCallWork: totalCallsAccepted > 0 ? (totalAfterCallWork /
totalCallsAccepted) : 0,
      onQueuePercent: metrics.length > 0 ? (totalOnQueuePercent /
metrics.length) : 0,
      qualityScore: 75.0 // Placeholder - would come from QualityService in
real implementation
    };

    // Mock data for trends analysis
    const trends = {
      busiestDay: 'Monday',
      busiestHour: '7pm',
      slowestDay: 'Saturday',
      slowestHour: '4am'
    };

    // Mock data for counselor performance
    const counselors = [
      {
        name: 'Jane Smith',
        team: 'Digital',
        answerRate: 96.5,
        avgTalkTime: 17.3,
        onQueuePercent: 88.7,
        callsPerShift: 6.2,
        qualityScore: 78.5
      },
      {
        name: 'John Doe',
        team: 'Lifeline',
        answerRate: 94.2,
        avgTalkTime: 19.8,
        onQueuePercent: 85.3,
```

```
        callsPerShift: 8.9,
        qualityScore: 72.1
      }
    ];

    // Return complete report
    return {
      success: true,
      summary: summary,
      trends: trends,
      counselors: counselors,
      detailed: metrics.map(metric => ({
        date: metric.date,
        callsOffered: metric.callsOffered,
        callsAccepted: metric.callsAccepted,
        answerRate: metric.answerRate,
        talkTime: metric.talkTime,
        afterCallWork: metric.afterCallWork,
        onQueuePercent: metric.onQueuePercent
      }))
    };
  } catch (error) {
    logError('getMetricsReport', error);
    return {
      success: false,
      message: 'Error generating metrics report: ' + error.message
    };
  }
},
 /**
 * Exports metrics report to a spreadsheet
 * @param {object} options - Options for filtering metrics
 * @return {object} Result of the operation
```

```
 */
exportMetricsReport: function(options = {}) {
  try {
    // Get metrics report data
    const reportData = this.getMetricsReport(options);

    if (!reportData.success) {
      return reportData;
    }

    const ss = SpreadsheetApp.getActiveSpreadsheet();

    // Create a new sheet for the report
    let reportSheet = ss.getSheetByName('Metrics Report');
    if (reportSheet) {
      ss.deleteSheet(reportSheet);
    }
    reportSheet = ss.insertSheet('Metrics Report');

    // Add report header
    const startDate = options.startDate ? new
Date(options.startDate).toLocaleDateString() : 'all time';
    const endDate = options.endDate ? new
Date(options.endDate).toLocaleDateString() : 'present';
    reportSheet.appendRow(['988 Lifeline LGBTQIA+ Services - Metrics
Report']);
    reportSheet.appendRow(['Period:', `${startDate} to ${endDate}`]);
    reportSheet.appendRow(['Generated:', new Date().toLocaleString()]);
    reportSheet.appendRow(['']);

    // Add summary section
    reportSheet.appendRow(['SUMMARY METRICS']);
    reportSheet.appendRow(['Metric', 'Value', 'Target']);
```

```
    reportSheet.appendRow(['Total Calls', reportData.summary.totalCalls,
''']);
    reportSheet.appendRow(['Answer Rate',
`${reportData.summary.answerRate.toFixed(1)}%`, '95%']);
    reportSheet.appendRow(['Average Talk Time',
`${reportData.summary.avgTalkTime.toFixed(2)} min`, '15-25 min']);
    reportSheet.appendRow(['After Call Work',
`${reportData.summary.afterCallWork.toFixed(2)} min`, '< 5 min']);
    reportSheet.appendRow(['On Queue %',
`${reportData.summary.onQueuePercent.toFixed(1)}%`, '85%']);
    reportSheet.appendRow(['Quality Score',
reportData.summary.qualityScore.toFixed(1), '70+']);
    reportSheet.appendRow(['']);

    // Format summary section

reportSheet.getRange('A5:C5').setFontWeight('bold').setBackground('#E0E0E0');

reportSheet.getRange('A6:C6').setFontWeight('bold').setBackground('#F3F3F3');

    // Add counselor performance section
    reportSheet.appendRow(['COUNSELOR PERFORMANCE']);
    reportSheet.appendRow(['Name', 'Team', 'Answer Rate', 'Avg Talk Time',
'On Queue %', 'Calls/Shift', 'Quality Score']);

    reportData.counselors.forEach(counselor => {
      reportSheet.appendRow([
        counselor.name,
        counselor.team,
        `${counselor.answerRate.toFixed(1)}%`,
        `${counselor.avgTalkTime.toFixed(2)} min`,
        `${counselor.onQueuePercent.toFixed(1)}%`,
        counselor.callsPerShift.toFixed(1),
```

```
        counselor.qualityScore.toFixed(1)
      ]);
    });

    reportSheet.appendRow(['']);

    // Format counselor section
reportSheet.getRange('A14:G14').setFontWeight('bold').setBackground('#E0E0E0')
;
reportSheet.getRange('A15:G15').setFontWeight('bold').setBackground('#F3F3F3')
;

    // Add detailed metrics section
    reportSheet.appendRow(['DETAILED METRICS']);
    reportSheet.appendRow(['Date', 'Calls Offered', 'Calls Accepted', 'Answer
Rate', 'Talk Time (min)', 'After Call Work (min)', 'On Queue %']);

    reportData.detailed.forEach(day => {
      reportSheet.appendRow([
        day.date instanceof Date ? day.date : new Date(day.date),
        day.callsOffered,
        day.callsAccepted,
        `${day.answerRate.toFixed(1)}%`,
        day.talkTime.toFixed(2),
        day.afterCallWork.toFixed(2),
        `${day.onQueuePercent.toFixed(1)}%`
      ]);
    });

    // Format detailed section
```

```
      const detailedStartRow = reportSheet.getLastRow() -
reportData.detailed.length - 1;

reportSheet.getRange(`A${detailedStartRow}:G${detailedStartRow}`).setFontWeigh
t('bold').setBackground('#E0E0E0');
      reportSheet.getRange(`A${detailedStartRow + 1}:G${detailedStartRow +
1}`).setFontWeight('bold').setBackground('#F3F3F3');

      // Auto-resize columns
      reportSheet.autoResizeColumns(1, 7);

      return {
        success: true,
        message: 'Metrics report exported successfully'
      };
    } catch (error) {
      logError('exportMetricsReport', error);
      return {
        success: false,
        message: 'Error exporting metrics report: ' + error.message
      };
    }
  }
};

// Expose functions to UI
function addDailyMetrics(metricsData) {
  return MetricsService.addDailyMetrics(metricsData);
}

function getMetrics(options) {
  return MetricsService.getMetrics(options);
}
```

```
function getMetricsReport(options) {
 return MetricsService.getMetricsReport(options);
}

function exportMetricsReport(options) {
 return MetricsService.exportMetricsReport(options);
}
/**
* OneOnOneService.gs
* Service for managing 1:1 meetings with counselors in the Crisis Support
System
*/
const OneOnOneService = {
 /**
  * Get all 1:1 meetings for a specific counselor
  * @param {string} counselorEmail - Email of the counselor
  * @return {Array} Array of 1:1 meeting objects
  */
 getOneOnOneMeetings: function(counselorEmail) {
   try {
     const ss = SpreadsheetApp.getActiveSpreadsheet();
     let sheet = ss.getSheetByName('1:1 Notes');

     if (!sheet) {
       // Return empty array if sheet doesn't exist
       return [];
     }

     // Get all data
     const data = sheet.getDataRange().getValues();

     // Skip header row
```

```
    if (data.length <= 1) {
      return [];
    }

    // Get column indices from headers
    const headers = data[0];
    const idIndex = headers.indexOf('ID');
    const dateIndex = headers.indexOf('Date');
    const emailIndex = headers.indexOf('Email');
    const counselorIndex = headers.indexOf('Counselor');
    const managerIndex = headers.indexOf('Manager');
    const summaryIndex = headers.indexOf('Summary');
    const nextCheckInIndex = headers.indexOf('Next Check-In Date');

    // Check if required columns exist
    if (dateIndex === -1 || emailIndex === -1) {
      console.error('Required columns not found in 1:1 Notes sheet');
      return [];
    }

    // Map data to meetings
    const meetings = [];

    for (let i = 1; i < data.length; i++) {
      const row = data[i];

      // Filter by counselor email
      if (row[emailIndex] && row[emailIndex].toLowerCase() ===
counselorEmail.toLowerCase()) {
        meetings.push({
          id: idIndex !== -1 ? row[idIndex] : `meeting_${i}`,
          meetingDate: dateIndex !== -1 ? row[dateIndex] : null,
          counselor: counselorIndex !== -1 ? row[counselorIndex] : '',
```

```
          email: row[emailIndex],
          manager: managerIndex !== -1 ? row[managerIndex] : '',
          summary: summaryIndex !== -1 ? row[summaryIndex] : '',
          nextCheckInDate: nextCheckInIndex !== -1 ? row[nextCheckInIndex] :
null,
        });
      }
    }

    return meetings;
  } catch (error) {
    logError('OneOnOneService.getOneOnOneMeetings', error);
    return [];
  }
},
 /**
 * Get details for a specific 1:1 meeting
 * @param {string} meetingId - ID of the meeting
 * @return {Object} Meeting details or null if not found
 */
getOneOnOneMeetingDetails: function(meetingId) {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    const sheet = ss.getSheetByName('1:1 Notes');

    if (!sheet) {
      return null;
    }

    // Get all data
    const data = sheet.getDataRange().getValues();

    // Get column indices from headers
```

```
    const headers = data[0];
    const idIndex = headers.indexOf('ID');

    // Find meeting by ID
    for (let i = 1; i < data.length; i++) {
      const row = data[i];

      if (idIndex !== -1 && row[idIndex] === meetingId) {
        // Create meeting object with all properties
        const meeting = {
          id: row[idIndex]
        };

        // Add all other fields
        for (let j = 0; j < headers.length; j++) {
          if (j !== idIndex) {
            meeting[this.camelCase(headers[j])] = row[j];
          }
        }

        // Parse JSON fields if they exist
        try {
          if (meeting.actionItems && typeof meeting.actionItems === 'string')
{
            meeting.actionItems = JSON.parse(meeting.actionItems);
          }
        } catch (e) {
          console.error('Failed to parse action items:', e);
          meeting.actionItems = [];
        }

        return meeting;
      }
```

```
    }

    return null;
  } catch (error) {
    logError('OneOnOneService.getOneOnOneMeetingDetails', error);
    return null;
  }
},
 /**
 * Save a new 1:1 meeting
 * @param {Object} meetingData - Data for the 1:1 meeting
 * @return {Object} Result of the operation
 */
saveOneOnOneNotes: function(meetingData) {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    let sheet = ss.getSheetByName('1:1 Notes');

    // Create sheet if it doesn't exist
    if (!sheet) {
      sheet = ss.insertSheet('1:1 Notes');
      sheet.appendRow([
        'ID',
        'Date',
        'Counselor',
        'Email',
        'Manager',
        'Summary',
        'Next Check-In Date',
        'Prep Review Previous',
        'Prep Check Metrics',
        'Prep Gather Updates',
        'Prep Discussion Points',
```

```
        'Data Review Calls',
        'Data Check Indicators',
        'Data Note Issues',
        'Data Identify Resources',
        'Preparation Notes',
        'Status Priority',
        'Workload Status',
        'Immediate Blockers',
        'Energy Level',
        'Quantitative Metrics',
        'Qualitative Impact',
        'Strategic Insights',
        'Skill Mastery',
        'Career Trajectory',
        'Development Goals',
        'Obstacle Identification',
        'Leadership Support',
        'Commitment Synthesis',
        'Momentum Action Items',
        'Momentum Ownership',
        'Momentum Follow Up',
        'Doc Finalize Notes',
        'Doc Clarify Points',
        'Doc Action Items',
        'Doc Follow Up Resources',
        'Action Items',
        'Support Follow Up',
        'Support Mentorship',
        'Support Resources',
        'Reflection Support',
        'Reflection Facilitation',
        'Reflection Mechanisms',
        'Dev Growth Areas',
```

```
      'Dev Strengths',
      'Dev Career',
      'Created By',
      'Created Date'
    ]);
    sheet.getRange(1, 1, 1, 49)
      .setFontWeight('bold')
      .setBackground('#E0E0E0');
  }

  // Convert meetingData to proper format
  const formattedData = this.formatMeetingData(meetingData);

  // Get current user info
  const currentUser = Session.getActiveUser().getEmail();

  // Generate a unique ID for the meeting
  const meetingId = Utilities.getUuid();

  // Convert action items to JSON string
  let actionItemsJson = '[]';
  try {
    if (meetingData.actionItems && meetingData.actionItems.length > 0) {
      actionItemsJson = JSON.stringify(meetingData.actionItems);
    }
  } catch (e) {
    console.error('Failed to stringify action items:', e);
  }

  // Prepare row data
  const rowData = [
    meetingId,                                    // ID
    new Date(meetingData.meetingDate),            // Date
```

```
      meetingData.counselorName,                    // Counselor
      meetingData.counselorEmail,                   // Email
      currentUser,                                  // Manager
      meetingData.summary || '',                    // Summary
      meetingData.nextCheckInDate ? new Date(meetingData.nextCheckInDate) :
'', // Next Check-In Date
      formattedData.prepReviewPrevious || false,    // Prep Review Previous
      formattedData.prepCheckMetrics || false,      // Prep Check Metrics
      formattedData.prepGatherUpdates || false,     // Prep Gather Updates
      formattedData.prepDiscussionPoints || false,  // Prep Discussion
Points
      formattedData.dataReviewCalls || false,       // Data Review Calls
      formattedData.dataCheckIndicators || false,   // Data Check Indicators
      formattedData.dataNoteIssues || false,        // Data Note Issues
      formattedData.dataIdentifyResources || false, // Data Identify
Resources
      formattedData.preparationNotes || '',         // Preparation Notes
      formattedData.statusPriority || '',           // Status Priority
      formattedData.workloadStatus || '',           // Workload Status
      formattedData.immediateBlockers || '',        // Immediate Blockers
      formattedData.energyLevel || '',              // Energy Level
      formattedData.quantitativeMetrics || '',      // Quantitative Metrics
      formattedData.qualitativeImpact || '',        // Qualitative Impact
      formattedData.strategicInsights || '',        // Strategic Insights
      formattedData.skillMastery || '',             // Skill Mastery
      formattedData.careerTrajectory || '',         // Career Trajectory
      formattedData.developmentGoals || '',         // Development Goals
      formattedData.obstacleIdentification || '',   // Obstacle
Identification
      formattedData.leadershipSupport || '',        // Leadership Support
      formattedData.commitmentSynthesis || '',      // Commitment Synthesis
      formattedData.momentumActionItems || false,   // Momentum Action Items
      formattedData.momentumOwnership || false,     // Momentum Ownership
```

```
      formattedData.momentumFollowUp || false,       // Momentum Follow Up
      formattedData.docFinalizeNotes || false,       // Doc Finalize Notes
      formattedData.docClarifyPoints || false,       // Doc Clarify Points
      formattedData.docActionItems || false,         // Doc Action Items
      formattedData.docFollowUpResources || false,   // Doc Follow Up
Resources
      actionItemsJson,                               // Action Items
      formattedData.supportFollowUp || false,        // Support Follow Up
      formattedData.supportMentorship || false,      // Support Mentorship
      formattedData.supportResources || false,       // Support Resources
      formattedData.reflectionSupport || '',         // Reflection Support
      formattedData.reflectionFacilitation || '',    // Reflection
Facilitation
      formattedData.reflectionMechanisms || '',      // Reflection Mechanisms
      formattedData.devGrowthAreas || '',            // Dev Growth Areas
      formattedData.devStrengths || '',              // Dev Strengths
      formattedData.devCareer || '',                 // Dev Career
      currentUser,                                   // Created By
      new Date()                                     // Created Date
    ];

    // Add to sheet
    sheet.appendRow(rowData);

    // Format date cells in the newly added row
    const lastRow = sheet.getLastRow();
    sheet.getRange(lastRow, 2).setNumberFormat('MM/dd/yyyy');
    sheet.getRange(lastRow, 7).setNumberFormat('MM/dd/yyyy');
    sheet.getRange(lastRow, 49).setNumberFormat('MM/dd/yyyy HH:mm:ss');

    // Log the activity
    logSystemActivity('1:1 Meeting', `Added 1:1 meeting for
${meetingData.counselorName}`);
```

```
    return {
      success: true,
      message: '1:1 meeting notes saved successfully',
      meetingId: meetingId
    };
  } catch (error) {
    logError('OneOnOneService.saveOneOnOneNotes', error);
    return {
      success: false,
      message: 'Error saving 1:1 meeting notes: ' + error.message
    };
  }
},
 /**
 * Send a follow-up email to a counselor after a 1:1 meeting
 * @param {Object} meetingData - Data for the 1:1 meeting
 * @return {Object} Result of the operation
 */
sendOneOnOneFollowUpEmail: function(meetingData) {
  try {
    // Validate required fields
    if (!meetingData.counselorEmail || !meetingData.counselorName) {
      return {
        success: false,
        message: 'Counselor email and name are required'
      };
    }

    // Extract action items for email
    let actionItemsHtml = '<p>No specific action items were recorded.</p>';

    if (meetingData.actionItems && meetingData.actionItems.length > 0) {
```

```
      actionItemsHtml = '<ul>';
      meetingData.actionItems.forEach(item => {
        const dueDate = item.date ? ` (due: ${new
Date(item.date).toLocaleDateString()})` : '';
        actionItemsHtml +=
`<li><strong>${item.description}</strong>${dueDate} - Owner:
${item.owner}</li>`;
      });
      actionItemsHtml += '</ul>';
    }

    // Get user's name for the email
    const userInfo = TeamService.getUserInfo();
    const managerName = userInfo.name || userInfo.email;

    // Set next check-in info
    let nextCheckInInfo = 'Not yet scheduled';
    if (meetingData.nextCheckInDate) {
      const nextDate = new Date(meetingData.nextCheckInDate);
      nextCheckInInfo = nextDate.toLocaleDateString();
    }

    // Create email subject and body
    const subject = `1:1 Meeting Notes - ${new
Date(meetingData.meetingDate).toLocaleDateString()}`;

    const htmlBody = `
      <div style="font-family: Arial, sans-serif; max-width: 600px; margin: 0
auto;">
        <h2 style="color: #001A4E;">1:1 Meeting Summary</h2>
        <p>Hello ${meetingData.counselorName},</p>
```

```
        <p>Thank you for our 1:1 meeting on ${new
Date(meetingData.meetingDate).toLocaleDateString()}. Below is a summary of our
discussion and action items.</p>

        <div style="margin: 20px 0; padding: 15px; background-color: #FFF2DF;
border-left: 4px solid #FF786E; border-radius: 4px;">
          <h3 style="margin-top: 0; color: #001A4E;">Meeting Summary</h3>
          <p>${meetingData.summary || 'We discussed your current work and
development opportunities.'}</p>
        </div>

        <div style="margin: 20px 0;">
          <h3 style="color: #001A4E;">Key Discussion Points</h3>
          <ul>
            ${meetingData.statusPriority ? `<li><strong>Priorities:</strong>
${meetingData.statusPriority}</li>` : ''}
            ${meetingData.strategicInsights ? `<li><strong>Strategic
Insights:</strong> ${meetingData.strategicInsights}</li>` : ''}
            ${meetingData.developmentGoals ? `<li><strong>Development
Goals:</strong> ${meetingData.developmentGoals}</li>` : ''}
          </ul>
        </div>

        <div style="margin: 20px 0;">
          <h3 style="color: #001A4E;">Action Items</h3>
          ${actionItemsHtml}
        </div>

        <div style="margin: 20px 0; padding: 15px; background-color: #BAE2CE;
border-radius: 4px;">
          <h3 style="margin-top: 0; color: #001A4E;">Next Steps</h3>
          <p><strong>Next Check-in:</strong> ${nextCheckInInfo}</p>
```

```
          <p>Please don't hesitate to reach out if you have any questions or need additional support before our next check-in.</p>
        </div>

        <p>Best regards,<br>${managerName}</p>
      </div>
    `;

    // Send the email
    GmailApp.sendEmail(
      meetingData.counselorEmail,
      subject,
      `Summary of our 1:1 meeting on ${new Date(meetingData.meetingDate).toLocaleDateString()}. Next check-in: ${nextCheckInInfo}`,
      {
        htmlBody: htmlBody,
        name: `${managerName} (Crisis Support System)`
      }
    );

    // Log the activity
    logSystemActivity('1:1 Meeting', `Sent follow-up email to ${meetingData.counselorName}`);

    return {
      success: true,
      message: 'Follow-up email sent successfully'
    };
  } catch (error) {
    logError('OneOnOneService.sendOneOnOneFollowUpEmail', error);
    return {
      success: false,
```

```
      message: 'Error sending follow-up email: ' + error.message
    };
  }
},
 /**
 * Format meeting data to ensure all required fields exist
 * @param {Object} meetingData - Raw meeting data from the form
 * @return {Object} Formatted meeting data
 */
formatMeetingData: function(meetingData) {
  // Create a copy of the data
  const formattedData = Object.assign({}, meetingData);

  // Ensure boolean fields are actual booleans
  const booleanFields = [
    'prepReviewPrevious', 'prepCheckMetrics', 'prepGatherUpdates',
'prepDiscussionPoints',
    'dataReviewCalls', 'dataCheckIndicators', 'dataNoteIssues',
'dataIdentifyResources',
    'momentumActionItems', 'momentumOwnership', 'momentumFollowUp',
    'docFinalizeNotes', 'docClarifyPoints', 'docActionItems',
'docFollowUpResources',
    'supportFollowUp', 'supportMentorship', 'supportResources'
  ];

  booleanFields.forEach(field => {
    formattedData[field] = formattedData[field] === true ||
formattedData[field] === 'true';
  });

  return formattedData;
},
 /**
```

```
  * Convert a header name to camelCase for JavaScript object properties
  * @param {string} header - Header name from spreadsheet
  * @return {string} camelCase version of the header
  */
 camelCase: function(header) {
   // Replace special characters and split into words
   const words = header.replace(/[^\w\s]/g, ' ').split(/\s+/);

   // Convert to camelCase
   return words.map((word, index) => {
     if (index === 0) {
       return word.toLowerCase();
     }
     return word.charAt(0).toUpperCase() + word.slice(1).toLowerCase();
   }).join('');
 }
};

// Expose functions to UI
function getOneOnOneMeetings(counselorEmail) {
 return OneOnOneService.getOneOnOneMeetings(counselorEmail);
}

function getOneOnOneMeetingDetails(meetingId) {
 return OneOnOneService.getOneOnOneMeetingDetails(meetingId);
}

function saveOneOnOneNotes(meetingData) {
 return OneOnOneService.saveOneOnOneNotes(meetingData);
}

function sendOneOnOneFollowUpEmail(meetingData) {
 return OneOnOneService.sendOneOnOneFollowUpEmail(meetingData);
```

```
}

/**
* Shows the one-on-one meeting notes UI
*/
function showOneOnOneNotes() {
 try {
   const html = HtmlService.createHtmlOutputFromFile('one-on-one')
     .setWidth(900)
     .setHeight(700)
     .setTitle('Counselor 1:1 Meeting Notes');

   SpreadsheetApp.getUi().showModalDialog(html, 'Counselor 1:1 Meeting
Notes');
   logSystemActivity('UI', 'Opened Counselor 1:1 Meeting Notes');
 } catch (error) {
   logError('showOneOnOneNotes', error);
   SpreadsheetApp.getUi().alert('Error opening Counselor 1:1 Meeting Notes: '
+ error.message);
 }
}
/**
* Service for managing quality monitoring
*/
const QualityService = {
 /**
  * Saves a quality review
  * @param {object} reviewData - Data for the quality review
  * @return {object} Result of the operation
  */
 saveQualityReview: function(reviewData) {
   try {
     const ss = SpreadsheetApp.getActiveSpreadsheet();
```

```
    let sheet = ss.getSheetByName('Quality Reviews');

    // Create sheet if it doesn't exist
    if (!sheet) {
      sheet = ss.insertSheet('Quality Reviews');
      sheet.appendRow([
        'ID',
        'Date',
        'Counselor',
        'Interaction ID',
        'Total Score',
        'Reviewer',
        'Review Date',
        'Status'
      ]);
      sheet.getRange(1, 1, 1, 8)
        .setFontWeight('bold')
        .setBackground('#E0E0E0');
    }

    // Generate a unique ID for the review
    const uuid = Utilities.getUuid();

    // Calculate total score
    const totalPoints = reviewData.scores.reduce((sum, score) => sum +
score.points, 0);
    const maxPoints = reviewData.scores.length * 2; // Assuming each
criterion has a max of 2 points
    const totalScore = Math.round((totalPoints / maxPoints) * 100);

    // Extract relevant data for the main sheet
    const rowData = [
      uuid,                                  // ID
```

```
      reviewData.date,                        // Date
      reviewData.counselorName,               // Counselor
      reviewData.interactionId,               // Interaction ID
      totalScore,                             // Total Score
      Session.getActiveUser().getEmail(),     // Reviewer
      new Date().toISOString().split('T')[0], // Review Date
      'Completed'                             // Status
    ];

    // Add row to main sheet
    sheet.appendRow(rowData);

    // Save full review data to document properties for retrieval
    // This is a workaround since the sheet doesn't have enough columns for all the detailed scores
    const properties = PropertiesService.getDocumentProperties();
    properties.setProperty('quality_review_' + uuid, JSON.stringify({
      id: uuid,
      ...reviewData,
      totalScore: totalScore,
      reviewedBy: Session.getActiveUser().getEmail(),
      reviewDate: new Date().toISOString().split('T')[0]
    }));

    // Log the activity
    logSystemActivity('Quality Monitoring', `Saved quality review for ${reviewData.counselorName} (Score: ${totalScore})`);

    return {
      success: true,
      message: 'Quality review saved successfully',
      reviewId: uuid,
      totalScore: totalScore
```

```
    };
  } catch (error) {
    logError('saveQualityReview', error);
    return {
      success: false,
      message: 'Error saving quality review: ' + error.message
    };
  }
},
 /**
 * Gets quality reviews for a specific counselor
 * @param {string} counselorEmail - Email of the counselor
 * @return {Array} Quality reviews data
 */
getQualityReviewsForCounselor: function(counselorEmail) {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    const sheet = ss.getSheetByName('Quality Reviews');

    // Return empty array if sheet doesn't exist
    if (!sheet) {
      return [];
    }

    // Get all quality reviews data
    const data = sheet.getDataRange().getValues();

    // Skip if only header row
    if (data.length <= 1) {
      return [];
    }

    // Find column indices
```

```
    const headers = data[0];
    const idIndex = headers.indexOf('ID');
    const dateIndex = headers.indexOf('Date');
    const counselorIndex = headers.indexOf('Counselor');
    const interactionIdIndex = headers.indexOf('Interaction ID');
    const scoreIndex = headers.indexOf('Total Score');
    const reviewerIndex = headers.indexOf('Reviewer');
    const reviewDateIndex = headers.indexOf('Review Date');

    // Find reviews for this counselor
    // Note: Currently using name matching, but in a real system would match on email
    // This is a limitation of the current data structure
    const reviews = [];

    for (let i = 1; i < data.length; i++) {
      // Using a simple string comparison for demonstration
      // In a production system, would use a more robust method
      if (data[i][counselorIndex] && data[i][counselorIndex].toString().toLowerCase().includes(counselorEmail.split('@')[0].toLowerCase())) {
        reviews.push({
          id: data[i][idIndex],
          date: data[i][dateIndex],
          interactionId: data[i][interactionIdIndex],
          score: data[i][scoreIndex],
          reviewer: data[i][reviewerIndex],
          reviewDate: data[i][reviewDateIndex]
        });
      }
    }

    return reviews;
```

```
  } catch (error) {
    logError('getQualityReviewsForCounselor', error);
    return [];
  }
},
 /**
 * Gets all quality reviews
 * @return {Array} All quality reviews
 */
getAllQualityReviews: function() {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    const sheet = ss.getSheetByName('Quality Reviews');

    // Return empty array if sheet doesn't exist
    if (!sheet) {
      return [];
    }

    // Get all quality reviews data
    const data = sheet.getDataRange().getValues();

    // Skip if only header row
    if (data.length <= 1) {
      return [];
    }

    // Find column indices
    const headers = data[0];
    const idIndex = headers.indexOf('ID');
    const dateIndex = headers.indexOf('Date');
    const counselorIndex = headers.indexOf('Counselor');
    const interactionIdIndex = headers.indexOf('Interaction ID');
```

```
    const scoreIndex = headers.indexOf('Total Score');
    const reviewerIndex = headers.indexOf('Reviewer');
    const reviewDateIndex = headers.indexOf('Review Date');

    // Map data to objects
    const reviews = [];

    for (let i = 1; i < data.length; i++) {
      reviews.push({
        id: data[i][idIndex],
        date: data[i][dateIndex],
        counselor: data[i][counselorIndex],
        interactionId: data[i][interactionIdIndex],
        score: data[i][scoreIndex],
        reviewer: data[i][reviewerIndex],
        reviewDate: data[i][reviewDateIndex]
      });
    }

    return reviews;
  } catch (error) {
    logError('getAllQualityReviews', error);
    return [];
  }
},
 /**
 * Gets quality review details
 * @param {string} reviewId - ID of the review
 * @return {object} Review details
 */
getQualityReviewDetails: function(reviewId) {
  try {
    // First try to get from document properties
```

```
    const properties = PropertiesService.getDocumentProperties();
    const reviewJson = properties.getProperty('quality_review_' + reviewId);

    if (reviewJson) {
      // Found review in document properties
      return JSON.parse(reviewJson);
    }

    // If not found in document properties, try to get basic info from the
sheet
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    const sheet = ss.getSheetByName('Quality Reviews');

    if (!sheet) {
      return null;
    }

    // Get all review data
    const data = sheet.getDataRange().getValues();

    // Skip if only header row
    if (data.length <= 1) {
      return null;
    }

    // Find column indices
    const headers = data[0];
    const idIndex = headers.indexOf('ID');

    // Find review by ID
    for (let i = 1; i < data.length; i++) {
      if (data[i][idIndex] === reviewId) {
        // Found the review, but only have basic data
```

```
        const basicData = {};

        // Map sheet columns to object properties
        headers.forEach((header, index) => {
          basicData[header.toString().replace(/\s+/g, '')] = data[i][index];
        });

        return basicData;
      }
    }

    // Review not found
    return null;
  } catch (error) {
    logError('getQualityReviewDetails', error);
    return null;
  }
},
 /**
 * Gets quality stats for a counselor
 * @param {string} counselorEmail - Email of the counselor
 * @return {object} Quality stats
 */
getQualityStatsForCounselor: function(counselorEmail) {
  try {
    const reviews = this.getQualityReviewsForCounselor(counselorEmail);

    if (reviews.length === 0) {
      return {
        count: 0,
        averageScore: 0,
        lastReviewDate: null,
        lastScore: 0,
```

```
        trend: 'neutral'
      };
    }

    // Calculate statistics
    const totalScore = reviews.reduce((sum, review) => sum + (review.score ||
0), 0);
    const averageScore = Math.round(totalScore / reviews.length);

    // Sort reviews by date (newest first)
    reviews.sort((a, b) => new Date(b.date) - new Date(a.date));

    const lastReviewDate = reviews[0].date;
    const lastScore = reviews[0].score;

    // Calculate trend (if at least 2 reviews)
    let trend = 'neutral';
    if (reviews.length >= 2) {
      const lastTwoScores = reviews.slice(0, 2).map(review => review.score ||
0);
      if (lastTwoScores[0] > lastTwoScores[1]) {
        trend = 'improving';
      } else if (lastTwoScores[0] < lastTwoScores[1]) {
        trend = 'declining';
      }
    }

    return {
      count: reviews.length,
      averageScore: averageScore,
      lastReviewDate: lastReviewDate,
      lastScore: lastScore,
      trend: trend
```

```
    };
  } catch (error) {
    logError('getQualityStatsForCounselor', error);
    return {
      count: 0,
      averageScore: 0,
      lastReviewDate: null,
      lastScore: 0,
      trend: 'neutral'
    };
  }
},
 /**
 * Gets quality stats for all counselors
 * @return {object} Quality stats by counselor
 */
getQualityStatsByTeam: function() {
  try {
    const reviews = this.getAllQualityReviews();

    if (reviews.length === 0) {
      return {
        teamAverage: 0,
        counselors: []
      };
    }

    // Group by counselor
    const counselorReviews = {};

    reviews.forEach(review => {
      const counselor = review.counselor;
      if (!counselorReviews[counselor]) {
```

```
        counselorReviews[counselor] = [];
      }
      counselorReviews[counselor].push(review);
    });

    // Calculate stats for each counselor
    const counselorStats = Object.keys(counselorReviews).map(counselor => {
      const counselorData = counselorReviews[counselor];
      const totalScore = counselorData.reduce((sum, review) => sum +
(review.score || 0), 0);
      const averageScore = Math.round(totalScore / counselorData.length);

      // Sort reviews by date (newest first)
      counselorData.sort((a, b) => new Date(b.date) - new Date(a.date));

      const lastReviewDate = counselorData[0].date;
      const lastScore = counselorData[0].score;

      // Calculate trend (if at least 2 reviews)
      let trend = 'neutral';
      if (counselorData.length >= 2) {
        const lastTwoScores = counselorData.slice(0, 2).map(review =>
review.score || 0);
        if (lastTwoScores[0] > lastTwoScores[1]) {
          trend = 'improving';
        } else if (lastTwoScores[0] < lastTwoScores[1]) {
          trend = 'declining';
        }
      }

      return {
        name: counselor,
        count: counselorData.length,
```

```
        averageScore: averageScore,
        lastReviewDate: lastReviewDate,
        lastScore: lastScore,
        trend: trend
      };
    });

    // Calculate team average
    const teamTotalScore = counselorStats.reduce((sum, stats) => sum +
stats.averageScore, 0);
    const teamAverage = Math.round(teamTotalScore / counselorStats.length);

    return {
      teamAverage: teamAverage,
      counselors: counselorStats
    };
  } catch (error) {
    logError('getQualityStatsByTeam', error);
    return {
      teamAverage: 0,
      counselors: []
    };
  }
 }
};

// Expose functions to UI
function saveQualityReview(reviewData) {
 return QualityService.saveQualityReview(reviewData);
}

function getQualityReviewsForCounselor(counselorEmail) {
 return QualityService.getQualityReviewsForCounselor(counselorEmail);
```

```
}

function getAllQualityReviews() {
 return QualityService.getAllQualityReviews();
}

function getQualityReviewDetails(reviewId) {
 return QualityService.getQualityReviewDetails(reviewId);
}

function getQualityStatsForCounselor(counselorEmail) {
 return QualityService.getQualityStatsForCounselor(counselorEmail);
}

function getQualityStatsByTeam() {
 return QualityService.getQualityStatsByTeam();
}
/**
* Service for managing schedule
*/
const ScheduleService = {
 /**
  * Adds a shift to the schedule
  * @param {object} shiftData - Data for the new shift
  * @return {object} Result of the operation
  */
 addShift: function(shiftData) {
   try {
     const ss = SpreadsheetApp.getActiveSpreadsheet();
     let sheet = ss.getSheetByName(CRISIS_SUPPORT_CONFIG.SHEETS.SCHEDULE);

     // Create sheet if it doesn't exist
     if (!sheet) {
```

```
      sheet = ss.insertSheet(CRISIS_SUPPORT_CONFIG.SHEETS.SCHEDULE);
      sheet.appendRow([
        'ID',
        'Date',
        'Shift Type',
        'Counselor Email',
        'Counselor Name',
        'Start Time',
        'End Time',
        'Status',
        'Notes',
        'Created By',
        'Created At'
      ]);
      sheet.getRange(1, 1, 1, 11)
        .setFontWeight('bold')
        .setBackground('#E0E0E0');
    }

    // Generate unique ID
    const shiftId = Utilities.getUuid();

    // Get counselor name
    let counselorName = '';
    try {
      const teamMembers = TeamService.getTeamMembers();
      const counselor = teamMembers.find(member => member.email ===
shiftData.counselorEmail);
      if (counselor) {
        counselorName = counselor.name;
      } else {
        counselorName = shiftData.counselorEmail.split('@')[0];
      }
```

```
    } catch (e) {
      // If team service fails, use email as name
      counselorName = shiftData.counselorEmail.split('@')[0];
    }

    // Calculate shift times
    let startTime = '';
    let endTime = '';

    const shiftDate = new Date(shiftData.date);

    if (shiftData.shiftType === 'Morning') {
      startTime = new Date(shiftDate);
      startTime.setHours(8, 0, 0);

      endTime = new Date(shiftDate);
      endTime.setHours(16, 0, 0);
    } else if (shiftData.shiftType === 'Afternoon') {
      startTime = new Date(shiftDate);
      startTime.setHours(16, 0, 0);

      endTime = new Date(shiftDate);
      endTime.setHours(23, 59, 59);
    } else if (shiftData.shiftType === 'Night') {
      startTime = new Date(shiftDate);
      startTime.setHours(0, 0, 0);

      endTime = new Date(shiftDate);
      endTime.setHours(8, 0, 0);
    }

    // Get current user
    const currentUser = Session.getActiveUser().getEmail();
```

```
    // Prepare row data
    const rowData = [
      shiftId,
      new Date(shiftData.date),
      shiftData.shiftType,
      shiftData.counselorEmail,
      counselorName,
      startTime,
      endTime,
      'Scheduled',
      shiftData.notes || '',
      currentUser,
      new Date()
    ];

    // Add to sheet
    sheet.appendRow(rowData);

    // Log activity
    logSystemActivity('Schedule Management', `Added ${shiftData.shiftType}
shift for ${counselorName} on ${shiftData.date}`);

    return {
      success: true,
      message: 'Shift added successfully',
      shiftId: shiftId
    };
  } catch (error) {
    logError('addShift', error);
    return {
      success: false,
      message: 'Error adding shift: ' + error.message
```

```
    };
  }
},
 /**
 * Gets shifts for a date range
 * @param {string} startDate - Start date (YYYY-MM-DD)
 * @param {string} endDate - End date (YYYY-MM-DD)
 * @return {Array} Shift data
 */
getShifts: function(startDate, endDate) {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    const sheet = ss.getSheetByName(CRISIS_SUPPORT_CONFIG.SHEETS.SCHEDULE);

    // Return empty array if sheet doesn't exist
    if (!sheet) {
      return [];
    }

    // Get all data
    const data = sheet.getDataRange().getValues();

    // Skip if only header row
    if (data.length <= 1) {
      return [];
    }

    // Convert string dates to Date objects for comparison
    const start = new Date(startDate);
    start.setHours(0, 0, 0, 0);

    const end = new Date(endDate);
    end.setHours(23, 59, 59, 999);
```

```
    // Get column indices
    const headers = data[0];
    const idIndex = headers.indexOf('ID');
    const dateIndex = headers.indexOf('Date');
    const shiftTypeIndex = headers.indexOf('Shift Type');
    const counselorEmailIndex = headers.indexOf('Counselor Email');
    const counselorNameIndex = headers.indexOf('Counselor Name');
    const startTimeIndex = headers.indexOf('Start Time');
    const endTimeIndex = headers.indexOf('End Time');
    const statusIndex = headers.indexOf('Status');
    const notesIndex = headers.indexOf('Notes');

    // Filter shifts by date range
    const shifts = [];

    for (let i = 1; i < data.length; i++) {
      const shiftDate = new Date(data[i][dateIndex]);
      shiftDate.setHours(0, 0, 0, 0);

      // Skip invalid dates or dates outside range
      if (isNaN(shiftDate.getTime()) || shiftDate < start || shiftDate > end)
{
        continue;
      }

      shifts.push({
        id: data[i][idIndex],
        date: data[i][dateIndex],
        shiftType: data[i][shiftTypeIndex],
        counselorEmail: data[i][counselorEmailIndex],
        counselorName: data[i][counselorNameIndex],
        startTime: data[i][startTimeIndex],
```

```
        endTime: data[i][endTimeIndex],
        status: data[i][statusIndex],
        notes: data[i][notesIndex]
      });
    }

    return shifts;
  } catch (error) {
    logError('getShifts', error);
    return [];
  }
},
 /**
 * Gets all shifts
 * @return {Array} All shifts
 */
getAllShifts: function() {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    const sheet = ss.getSheetByName(CRISIS_SUPPORT_CONFIG.SHEETS.SCHEDULE);

    // Return empty array if sheet doesn't exist
    if (!sheet) {
      return [];
    }

    // Get all data
    const data = sheet.getDataRange().getValues();

    // Skip if only header row
    if (data.length <= 1) {
      return [];
    }
```

```
    // Get column indices
    const headers = data[0];
    const idIndex = headers.indexOf('ID');
    const dateIndex = headers.indexOf('Date');
    const shiftTypeIndex = headers.indexOf('Shift Type');
    const counselorEmailIndex = headers.indexOf('Counselor Email');
    const counselorNameIndex = headers.indexOf('Counselor Name');
    const startTimeIndex = headers.indexOf('Start Time');
    const endTimeIndex = headers.indexOf('End Time');
    const statusIndex = headers.indexOf('Status');
    const notesIndex = headers.indexOf('Notes');

    // Map all shifts
    const shifts = [];

    for (let i = 1; i < data.length; i++) {
      shifts.push({
        id: data[i][idIndex],
        date: data[i][dateIndex],
        shiftType: data[i][shiftTypeIndex],
        counselorEmail: data[i][counselorEmailIndex],
        counselorName: data[i][counselorNameIndex],
        startTime: data[i][startTimeIndex],
        endTime: data[i][endTimeIndex],
        status: data[i][statusIndex],
        notes: data[i][notesIndex]
      });
    }

    return shifts;
  } catch (error) {
    logError('getAllShifts', error);
```

```
    return [];
  }
},
 /**
 * Updates a shift
 * @param {object} updateData - Data for the shift update
 * @return {object} Result of the operation
 */
updateShift: function(updateData) {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    const sheet = ss.getSheetByName(CRISIS_SUPPORT_CONFIG.SHEETS.SCHEDULE);

    // Return error if sheet doesn't exist
    if (!sheet) {
      return {
        success: false,
        message: 'Schedule sheet not found'
      };
    }

    // Get all data
    const data = sheet.getDataRange().getValues();

    // Get column indices
    const headers = data[0];
    const idIndex = headers.indexOf('ID');
    const statusIndex = headers.indexOf('Status');
    const notesIndex = headers.indexOf('Notes');

    // Find the shift
    let shiftRow = -1;
```

```
    for (let i = 1; i < data.length; i++) {
      if (data[i][idIndex] === updateData.id) {
        shiftRow = i + 1; // +1 because sheet rows are 1-indexed
        break;
      }
    }

    // Return error if shift not found
    if (shiftRow === -1) {
      return {
        success: false,
        message: 'Shift not found'
      };
    }

    // Update the sheet
    if (updateData.status) {
      sheet.getRange(shiftRow, statusIndex + 1).setValue(updateData.status);
    }

    if (updateData.notes !== undefined) {
      sheet.getRange(shiftRow, notesIndex + 1).setValue(updateData.notes);
    }

    // Log activity
    logSystemActivity('Schedule Management', `Updated shift ${updateData.id}
status to ${updateData.status}`);

    return {
      success: true,
      message: 'Shift updated successfully'
    };
  } catch (error) {
```

```
    logError('updateShift', error);
    return {
      success: false,
      message: 'Error updating shift: ' + error.message
    };
  }
},
 /**
 * Deletes a shift
 * @param {string} shiftId - ID of the shift to delete
 * @return {object} Result of the operation
 */
deleteShift: function(shiftId) {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    const sheet = ss.getSheetByName(CRISIS_SUPPORT_CONFIG.SHEETS.SCHEDULE);

    // Return error if sheet doesn't exist
    if (!sheet) {
      return {
        success: false,
        message: 'Schedule sheet not found'
      };
    }

    // Get all data
    const data = sheet.getDataRange().getValues();

    // Get column indices
    const headers = data[0];
    const idIndex = headers.indexOf('ID');

    // Find the shift
```

```
    let shiftRow = -1;

    for (let i = 1; i < data.length; i++) {
      if (data[i][idIndex] === shiftId) {
        shiftRow = i + 1; // +1 because sheet rows are 1-indexed
        break;
      }
    }

    // Return error if shift not found
    if (shiftRow === -1) {
      return {
        success: false,
        message: 'Shift not found'
      };
    }

    // Delete the row
    sheet.deleteRow(shiftRow);

    // Log activity
    logSystemActivity('Schedule Management', `Deleted shift ${shiftId}`);

    return {
      success: true,
      message: 'Shift deleted successfully'
    };
  } catch (error) {
    logError('deleteShift', error);
    return {
      success: false,
      message: 'Error deleting shift: ' + error.message
    };
```

```
  }
},
 /**
 * Initializes a week with standard schedule
 * @param {object} weekData - Data for the week initialization
 * @return {object} Result of the operation
 */
initializeWeek: function(weekData) {
  try {
    // Validate start date
    if (!weekData.startDate) {
      return {
        success: false,
        message: 'Start date is required'
      };
    }

    const startDate = new Date(weekData.startDate);

    // Make sure start date is a Sunday
    if (startDate.getDay() !== 0) {
      return {
        success: false,
        message: 'Start date must be a Sunday'
      };
    }

    // Define standard template
    let template = this.getStandardTemplate();

    // Use custom template if specified
    if (weekData.templateType === 'custom' && weekData.counselorAssignments)
{
```

```
      template = this.getCustomTemplate(weekData.counselorAssignments);
    }

    // Initialize each day of the week
    const successfulShifts = [];
    const failedShifts = [];

    for (let dayOffset = 0; dayOffset < 7; dayOffset++) {
      const currentDate = new Date(startDate);
      currentDate.setDate(startDate.getDate() + dayOffset);

      // Get day name (0 = Sunday, 1 = Monday, etc.)
      const dayName = ['sunday', 'monday', 'tuesday', 'wednesday',
'thursday', 'friday', 'saturday'][dayOffset];

      // Process each shift type
      ['Morning', 'Afternoon', 'Night'].forEach(shiftType => {
        // Get counselor for this day and shift
        const shiftKey = `${dayName}${shiftType}`;
        const counselorEmail = template[shiftKey];

        // Skip if no counselor assigned
        if (!counselorEmail) {
          return;
        }

        // Add the shift
        const shiftData = {
          date: currentDate.toISOString().split('T')[0],
          shiftType: shiftType,
          counselorEmail: counselorEmail,
          notes: 'Auto-generated from weekly template'
        };
```

```
        const result = this.addShift(shiftData);

        if (result.success) {
          successfulShifts.push({
            day: dayName,
            shift: shiftType,
            counselor: counselorEmail
          });
        } else {
          failedShifts.push({
            day: dayName,
            shift: shiftType,
            counselor: counselorEmail,
            error: result.message
          });
        }
      });
    }

    // Log activity
    logSystemActivity('Schedule Management', `Initialized week starting
${weekData.startDate} (${successfulShifts.length} shifts added)`);

    return {
      success: true,
      message: `Week initialized with ${successfulShifts.length} shifts`,
      successfulShifts: successfulShifts,
      failedShifts: failedShifts
    };
  } catch (error) {
    logError('initializeWeek', error);
    return {
```

```
      success: false,
      message: 'Error initializing week: ' + error.message
    };
  }
},
 /**
 * Gets the standard schedule template
 * @return {object} Template with shift assignments
 */
getStandardTemplate: function() {
  // This is a placeholder for a standard template
  // In a real implementation, this might be stored in PropertiesService or
in a sheet

  return {
    // Monday
    mondayMorning: 'counselor1@example.com',
    mondayAfternoon: 'counselor2@example.com',
    mondayNight: 'counselor3@example.com',

    // Tuesday
    tuesdayMorning: 'counselor4@example.com',
    tuesdayAfternoon: 'counselor1@example.com',
    tuesdayNight: 'counselor2@example.com',

    // Wednesday
    wednesdayMorning: 'counselor3@example.com',
    wednesdayAfternoon: 'counselor4@example.com',
    wednesdayNight: 'counselor1@example.com',

    // Thursday
    thursdayMorning: 'counselor2@example.com',
    thursdayAfternoon: 'counselor3@example.com',
```

```
    thursdayNight: 'counselor4@example.com',

    // Friday
    fridayMorning: 'counselor1@example.com',
    fridayAfternoon: 'counselor2@example.com',
    fridayNight: 'counselor3@example.com',

    // Weekend shifts can be left empty if not staffed
    saturdayMorning: '',
    saturdayAfternoon: 'counselor4@example.com',
    saturdayNight: 'counselor1@example.com',

    sundayMorning: '',
    sundayAfternoon: 'counselor2@example.com',
    sundayNight: 'counselor3@example.com'
  };
},
 /**
 * Creates a custom template from assignments
 * @param {object} assignments - Custom shift assignments
 * @return {object} Template with shift assignments
 */
getCustomTemplate: function(assignments) {
  // Start with a template where all shifts are unassigned
  const template = {
    // Monday
    mondayMorning: '',
    mondayAfternoon: '',
    mondayNight: '',

    // Tuesday
    tuesdayMorning: '',
    tuesdayAfternoon: '',
```

```
    tuesdayNight: '',

    // Wednesday
    wednesdayMorning: '',
    wednesdayAfternoon: '',
    wednesdayNight: '',

    // Thursday
    thursdayMorning: '',
    thursdayAfternoon: '',
    thursdayNight: '',

    // Friday
    fridayMorning: '',
    fridayAfternoon: '',
    fridayNight: '',

    // Weekend
    saturdayMorning: '',
    saturdayAfternoon: '',
    saturdayNight: '',

    sundayMorning: '',
    sundayAfternoon: '',
    sundayNight: ''
  };

  // Apply custom assignments
  if (assignments.mondayMorning) template.mondayMorning =
assignments.mondayMorning;
  if (assignments.mondayAfternoon) template.mondayAfternoon =
assignments.mondayAfternoon;
```

```
    if (assignments.mondayNight) template.mondayNight =
assignments.mondayNight;

    // In a real implementation, this would include all days and shifts

    return template;
  }
};

// Expose functions to UI
function addShift(shiftData) {
  return ScheduleService.addShift(shiftData);
}

function getShifts(startDate, endDate) {
  return ScheduleService.getShifts(startDate, endDate);
}

function getAllShifts() {
  return ScheduleService.getAllShifts();
}

function updateShift(updateData) {
  return ScheduleService.updateShift(updateData);
}

function deleteShift(shiftId) {
  return ScheduleService.deleteShift(shiftId);
}

function initializeWeek(weekData) {
  return ScheduleService.initializeWeek(weekData);
}
```

```
/**
* Service for managing system settings and configuration
*/
const SettingsService = {
/**
* Initialize system properties with default or existing values
* @return {object} Result of initialization
*/
initializeSystemProperties: function() {
 try {
   const props = PropertiesService.getScriptProperties();

   const requiredProperties = {
     [CRISIS_SUPPORT_CONFIG.ASANA.API_KEY_PROPERTY]: '',
     [CRISIS_SUPPORT_CONFIG.ASANA.WORKSPACE_GID_PROPERTY]: '',
     [CRISIS_SUPPORT_CONFIG.ASANA.PROJECT_GID_PROPERTY]: '',
     'systemInitialized': 'false',
     'initializationDate': '',
     'organizationName': '988 Lifeline LGBTQIA+ Services',
     'primaryContact': '',
     'timeZone': 'America/New_York',
     'alertEmailRecipients': 'support@trevorproject.org'
   };

   let propertiesUpdated = false;

   Object.entries(requiredProperties).forEach(([key, defaultValue]) => {
     const existingValue = props.getProperty(key);

     if (!existingValue) {
       props.setProperty(key, defaultValue);
       propertiesUpdated = true;
     }
```

```
    });

    // Log initialization if properties were updated
    if (propertiesUpdated) {
      logSystemActivity('Settings', 'Initialized system properties');
    }

    return {
      success: true,
      message: 'System properties initialized',
      propertiesUpdated: propertiesUpdated
    };
  } catch (error) {
    logError('initializeSystemProperties', error);
    return {
      success: false,
      message: 'Error initializing system properties: ' + error.message
    };
  }
},

/**
 * Interactive system configuration setup
 * @return {object} Result of configuration setup
 */
setupSystemConfiguration: function() {
  try {
    // Initialize properties first
    this.initializeSystemProperties();

    const ui = SpreadsheetApp.getUi();

    // Configuration prompts
```

```
    const configurations = [
      {
        title: 'Organization Name',
        property: 'organizationName',
        prompt: 'Enter your organization name:'
      },
      {
        title: 'Asana API Configuration',
        property: 'asanaApiKey',
        prompt: 'Enter your Asana API Key:'
      },
      {
        title: 'Asana Workspace Configuration',
        property: 'asanaWorkspaceGid',
        prompt: 'Enter your Asana Workspace GID:'
      },
      {
        title: 'Asana Project Configuration',
        property: 'asanaProjectGid',
        prompt: 'Enter your Asana Project GID:'
      },
      {
        title: 'Alert Email Recipients',
        property: 'alertEmailRecipients',
        prompt: 'Enter email addresses for alert notifications
(comma-separated):'
      }
    ];

    const props = PropertiesService.getScriptProperties();

    configurations.forEach(config => {
      const response = ui.prompt(
```

```
        config.title,
        config.prompt,
        ui.ButtonSet.OK_CANCEL
      );

      if (response.getSelectedButton() == ui.Button.OK) {
        const value = response.getResponseText().trim();
        props.setProperty(config.property, value);
      }
    });

    // Mark system as initialized
    props.setProperty('systemInitialized', 'true');
    props.setProperty('initializationDate', new Date().toISOString());

    // Log activity
    logSystemActivity('Settings', 'Completed system configuration');

    ui.alert('System Configuration', 'System configuration completed
successfully!', ui.ButtonSet.OK);

    return {
      success: true,
      message: 'System configuration completed'
    };
  } catch (error) {
    logError('setupSystemConfiguration', error);
    SpreadsheetApp.getUi().alert('Configuration Error', error.message,
SpreadsheetApp.getUi().ButtonSet.OK);

    return {
      success: false,
      message: 'Error during system configuration: ' + error.message
```

```
    };
  }
},

// ... (rest of the existing methods remain the same)

/**
* Comprehensive system configuration validation
* @return {object} Validation results
*/
validateSystemConfiguration: function() {
 const props = PropertiesService.getScriptProperties();
 const missingProperties = [];

 // Explicitly check Asana-specific properties
 const asanaProperties = [
   {
     key: CRISIS_SUPPORT_CONFIG.ASANA.API_KEY_PROPERTY,
     name: 'Asana API Key'
   },
   {
     key: CRISIS_SUPPORT_CONFIG.ASANA.WORKSPACE_GID_PROPERTY,
     name: 'Asana Workspace GID'
   },
   {
     key: CRISIS_SUPPORT_CONFIG.ASANA.PROJECT_GID_PROPERTY,
     name: 'Asana Project GID'
   }
 ];

 // Check each Asana property
 asanaProperties.forEach(prop => {
   const value = props.getProperty(prop.key);
```

```
    if (!value || value.trim() === '') {
      missingProperties.push(prop.name);
    }
  });

  // Additional configuration checks
  if (!props.getProperty('systemInitialized')) {
    missingProperties.push('System Initialization');
  }

  if (!props.getProperty('initializationDate')) {
    missingProperties.push('Initialization Date');
  }

  return {
    isConfigured: missingProperties.length === 0,
    missingProperties: missingProperties
  };
},

/**
 * Get comprehensive system configuration
 * @return {object} Complete system configuration
 */
getSystemConfiguration: function() {
  const props = PropertiesService.getScriptProperties();

  return {
    systemInitialized: props.getProperty('systemInitialized') === 'true',
    initializationDate: props.getProperty('initializationDate'),
    generalSettings: {
      organizationName: props.getProperty('organizationName'),
      primaryContact: props.getProperty('primaryContact'),
```

```
      timeZone: props.getProperty('timeZone')
    },
    asanaSettings: {
      apiKey:
props.getProperty(CRISIS_SUPPORT_CONFIG.ASANA.API_KEY_PROPERTY),
      workspaceGid:
props.getProperty(CRISIS_SUPPORT_CONFIG.ASANA.WORKSPACE_GID_PROPERTY),
      projectGid:
props.getProperty(CRISIS_SUPPORT_CONFIG.ASANA.PROJECT_GID_PROPERTY)
    },
    alertEmailRecipients: props.getProperty('alertEmailRecipients')
  };
 }
};

// Expose functions to UI
function initializeSystemProperties() {
 return SettingsService.initializeSystemProperties();
}

function setupSystemConfiguration() {
 return SettingsService.setupSystemConfiguration();
}

function validateSystemConfiguration() {
 const result = SettingsService.validateSystemConfiguration();
 const ui = SpreadsheetApp.getUi();
  if (result.isConfigured) {
   ui.alert('Configuration Status', 'System is fully configured!',
ui.ButtonSet.OK);
 } else {
   ui.alert('Configuration Status',
```

```
    'Missing configuration properties:\n' +
result.missingProperties.join('\n'),
    ui.ButtonSet.OK
   );
 }
  return result;
}
/**
* ShiftService.gs
*
* Service for managing team lead shifts
*/
const ShiftService = {
 /**
  * Initializes a new shift
  * @param {object} shiftData - Shift data
  * @return {object} Result of the operation
  */
 initializeShift: function(shiftData) {
   try {
     const ss = SpreadsheetApp.getActiveSpreadsheet();
     let sheet = ss.getSheetByName('Team Lead Shifts');

     // Create sheet if it doesn't exist
     if (!sheet) {
       sheet = ss.insertSheet('Team Lead Shifts');
       sheet.appendRow([
         'Date',
         'Shift Type',
         'Start Time',
         'End Time',
         'Goals',
         'Quick Notes',
```

```
        'Created By',
        'Timestamp'
      ]);
      sheet.getRange(1, 1, 1, 8)
        .setFontWeight('bold')
        .setBackground('#E0E0E0');
    }

    // Validate required fields
    if (!shiftData.date || !shiftData.shiftType) {
      return {
        success: false,
        message: 'Date and shift type are required'
      };
    }

    // Format date and times
    const shiftDate = shiftData.date instanceof Date ? shiftData.date : new
Date(shiftData.date);
    const startTime = shiftData.startTime || '';
    const endTime = shiftData.endTime || '';

    // Get user info
    const user = Session.getActiveUser().getEmail();

    // Prepare row data
    const rowData = [
      shiftDate,
      shiftData.shiftType,
      startTime,
      endTime,
      shiftData.goals || '',
      shiftData.quickNotes || '',
```

```
      user,
      new Date()
    ];

    // Add to sheet
    sheet.appendRow(rowData);

    // Log activity
    logSystemActivity('Shift Management', `Initialized ${shiftData.shiftType}
shift for ${Utilities.formatDate(shiftDate, Session.getScriptTimeZone(),
'yyyy-MM-dd')}`);

    return {
      success: true,
      message: 'Shift initialized successfully',
      shiftData: {
        date: shiftDate,
        shiftType: shiftData.shiftType,
        startTime: startTime,
        endTime: endTime,
        goals: shiftData.goals,
        quickNotes: shiftData.quickNotes
      }
    };
  } catch (error) {
    logError('initializeShift', error);
    return {
      success: false,
      message: 'Error initializing shift: ' + error.message
    };
  }
},
 /**
```

```
 * Gets shift data for a specific date or the most recent shift
 * @param {object} options - Options for filtering shifts
 * @return {object} Shift data
 */
getShiftData: function(options = {}) {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    const sheet = ss.getSheetByName('Team Lead Shifts');

    if (!sheet) {
      return {
        success: false,
        message: 'Team Lead Shifts sheet not found'
      };
    }

    // Get all data
    const data = sheet.getDataRange().getValues();

    // Skip header row
    if (data.length <= 1) {
      return {
        success: false,
        message: 'No shift data found'
      };
    }

    // Get column indices
    const headers = data[0];
    const dateIndex = headers.indexOf('Date');
    const shiftTypeIndex = headers.indexOf('Shift Type');
    const startTimeIndex = headers.indexOf('Start Time');
    const endTimeIndex = headers.indexOf('End Time');
```

```
    const goalsIndex = headers.indexOf('Goals');
    const quickNotesIndex = headers.indexOf('Quick Notes');
    const createdByIndex = headers.indexOf('Created By');

    // Check if required columns exist
    if (dateIndex === -1 || shiftTypeIndex === -1) {
      return {
        success: false,
        message: 'Required columns not found in Team Lead Shifts sheet'
      };
    }

    // Filter by date if provided
    let filteredData = data.slice(1);

    if (options.date) {
      const targetDate = options.date instanceof Date ? options.date : new
Date(options.date);
      const targetDateString = Utilities.formatDate(targetDate,
Session.getScriptTimeZone(), 'yyyy-MM-dd');

      filteredData = filteredData.filter(row => {
        const rowDate = row[dateIndex];
        if (!rowDate) return false;

        const rowDateString = Utilities.formatDate(rowDate,
Session.getScriptTimeZone(), 'yyyy-MM-dd');
        return rowDateString === targetDateString;
      });
    }

    // Filter by user if provided
    if (options.user) {
```

```
      filteredData = filteredData.filter(row => row[createdByIndex] ===
options.user);
    }

    // Sort by date (newest first) if looking for most recent
    if (options.mostRecent) {
      filteredData.sort((a, b) => {
        if (!a[dateIndex]) return 1;
        if (!b[dateIndex]) return -1;
        return b[dateIndex] - a[dateIndex];
      });
    }

    // Return the first matching shift (or most recent)
    if (filteredData.length > 0) {
      const shift = filteredData[0];

      return {
        success: true,
        found: true,
        shift: {
          date: shift[dateIndex],
          shiftType: shift[shiftTypeIndex],
          startTime: shift[startTimeIndex],
          endTime: shift[endTimeIndex],
          goals: shift[goalsIndex],
          quickNotes: shift[quickNotesIndex],
          createdBy: shift[createdByIndex]
        }
      };
    } else {
      return {
        success: true,
```

```
        found: false,
        message: 'No matching shift found'
      };
    }
  } catch (error) {
    logError('getShiftData', error);
    return {
      success: false,
      message: 'Error retrieving shift data: ' + error.message
    };
  }
 }
};

// Expose functions to UI
function initializeShift(shiftData) {
 return ShiftService.initializeShift(shiftData);
}

function getShiftData(options) {
 return ShiftService.getShiftData(options);
}
/**
* TaskService.gs
*
* Service for managing tasks functionality
*/
const TaskService = {
 /**
  * Creates a new task
  * @param {object} taskData - Task data
  * @return {object} Result of the operation
  */
```

```
createTask: function(taskData) {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    let sheet = ss.getSheetByName(CRISIS_SUPPORT_CONFIG.SHEETS.TASKS);

    // Create sheet if it doesn't exist
    if (!sheet) {
      sheet = ss.insertSheet(CRISIS_SUPPORT_CONFIG.SHEETS.TASKS);
      sheet.appendRow([
        'Date Created',
        'Task',
        'Description',
        'Assignee',
        'Due Date',
        'Priority',
        'Status',
        'Completed Date',
        'Completed By'
      ]);
      sheet.getRange(1, 1, 1, 9)
        .setFontWeight('bold')
        .setBackground('#E0E0E0');
    }

    // Validate required fields
    if (!taskData.task) {
      return {
        success: false,
        message: 'Task name is required'
      };
    }

    // Prepare row data
```

```
    const rowData = [
      new Date(),
      taskData.task,
      taskData.description || '',
      taskData.assignee || '',
      taskData.dueDate instanceof Date ? taskData.dueDate : (taskData.dueDate
? new Date(taskData.dueDate) : ''),
      taskData.priority || 'Medium',
      'Open',
      '',
      ''
    ];

    // Add to sheet
    sheet.appendRow(rowData);

    // Log activity
    logSystemActivity('Task Management', `Created task: ${taskData.task}`);

    return {
      success: true,
      message: 'Task created successfully'
    };
  } catch (error) {
    logError('createTask', error);
    return {
      success: false,
      message: 'Error creating task: ' + error.message
    };
  }
},
 /**
 * Creates a new task in Asana
```

```
 * @param {object} taskData - Task data
 * @return {object} Result of the operation
 */
createAsanaTask: function(taskData) {
  try {
    // First create the task in our system
    const result = this.createTask(taskData);

    if (!result.success) {
      return result;
    }

    // Get Asana configuration
    const apiKey =
PropertiesService.getScriptProperties().getProperty(CRISIS_SUPPORT_CONFIG.ASAN
A.API_KEY_PROPERTY);
    const workspaceGid =
PropertiesService.getScriptProperties().getProperty(CRISIS_SUPPORT_CONFIG.ASAN
A.WORKSPACE_GID_PROPERTY);
    const projectGid =
PropertiesService.getScriptProperties().getProperty(CRISIS_SUPPORT_CONFIG.ASAN
A.PROJECT_GID_PROPERTY);

    if (!apiKey || !workspaceGid || !projectGid) {
      return {
        success: false,
        message: 'Asana configuration is incomplete. Please check settings.'
      };
    }

    // Mock Asana integration (in a real implementation, this would make API
calls to Asana)
    // This is just a placeholder to simulate the functionality
```

```
    // Log activity
    logSystemActivity('Task Management', `Created Asana task:
${taskData.task}`);

    return {
      success: true,
      message: 'Task created in Asana successfully',
      asanaLink: 'https://app.asana.com/0/' + projectGid + '/some-task-id'
    };
  } catch (error) {
    logError('createAsanaTask', error);
    return {
      success: false,
      message: 'Error creating Asana task: ' + error.message
    };
  }
},
 /**
 * Gets tasks from the system
 * @param {object} options - Options for filtering tasks
 * @return {Array} Tasks data
 */
getTasks: function(options = {}) {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    const sheet = ss.getSheetByName(CRISIS_SUPPORT_CONFIG.SHEETS.TASKS);

    if (!sheet) {
      return [];
    }

    // Get all data
```

```
    const data = sheet.getDataRange().getValues();

    // Skip header row
    if (data.length <= 1) {
      return [];
    }

    // Get column indices
    const headers = data[0];
    const dateCreatedIndex = headers.indexOf('Date Created');
    const taskIndex = headers.indexOf('Task');
    const descriptionIndex = headers.indexOf('Description');
    const assigneeIndex = headers.indexOf('Assignee');
    const dueDateIndex = headers.indexOf('Due Date');
    const priorityIndex = headers.indexOf('Priority');
    const statusIndex = headers.indexOf('Status');
    const completedDateIndex = headers.indexOf('Completed Date');
    const completedByIndex = headers.indexOf('Completed By');

    // Check if required columns exist
    if (taskIndex === -1 || statusIndex === -1) {
      throw new Error('Required columns not found in Tasks sheet');
    }

    // Map data to task objects
    let tasks = [];

    for (let i = 1; i < data.length; i++) {
      const row = data[i];

      // Skip empty rows
      if (!row[taskIndex]) {
        continue;
```

```
      }

      const task = {
        dateCreated: dateCreatedIndex !== -1 ? row[dateCreatedIndex] : '',
        task: row[taskIndex],
        description: descriptionIndex !== -1 ? row[descriptionIndex] || '' :
'',
        assignee: assigneeIndex !== -1 ? row[assigneeIndex] || '' : '',
        dueDate: dueDateIndex !== -1 ? row[dueDateIndex] || '' : '',
        priority: priorityIndex !== -1 ? row[priorityIndex] || 'Medium' :
'Medium',
        status: statusIndex !== -1 ? row[statusIndex] || 'Open' : 'Open',
        completedDate: completedDateIndex !== -1 ? row[completedDateIndex] ||
'' : '',
        completedBy: completedByIndex !== -1 ? row[completedByIndex] || '' :
'',
        rowIndex: i + 1 // Store the row index for later updates
      };

      tasks.push(task);
    }

    // Filter by status if specified
    if (options.status) {
      tasks = tasks.filter(task => task.status.toLowerCase() ===
options.status.toLowerCase());
    }

    // Filter by assignee if specified
    if (options.assignee) {
      tasks = tasks.filter(task => task.assignee.toLowerCase() ===
options.assignee.toLowerCase());
    }
```

```
    // Sort by due date (if present)
    tasks.sort((a, b) => {
      // Put tasks without due dates at the end
      if (!a.dueDate) return 1;
      if (!b.dueDate) return -1;

      // Sort by due date
      return new Date(a.dueDate) - new Date(b.dueDate);
    });

    return tasks;
  } catch (error) {
    logError('getTasks', error);
    return [];
  }
},
 /**
 * Completes a task
 * @param {number} rowIndex - Row index of the task
 * @param {string} completedBy - Email of user completing the task
 * @return {object} Result of the operation
 */
completeTask: function(rowIndex, completedBy) {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    const sheet = ss.getSheetByName(CRISIS_SUPPORT_CONFIG.SHEETS.TASKS);

    if (!sheet) {
      return {
        success: false,
        message: 'Tasks sheet not found'
      };
```

```
    }

    // Get column indices
    const headers = sheet.getRange(1, 1, 1,
sheet.getLastColumn()).getValues()[0];
    const taskIndex = headers.indexOf('Task');
    const statusIndex = headers.indexOf('Status');
    const completedDateIndex = headers.indexOf('Completed Date');
    const completedByIndex = headers.indexOf('Completed By');

    // Check if required columns exist
    if (statusIndex === -1 || completedDateIndex === -1 || completedByIndex
=== -1) {
      return {
        success: false,
        message: 'Required columns not found in Tasks sheet'
      };
    }

    // Get task name for logging
    const taskName = sheet.getRange(rowIndex, taskIndex + 1).getValue();

    // Update status, completed date, and completed by
    sheet.getRange(rowIndex, statusIndex + 1).setValue('Completed');
    sheet.getRange(rowIndex, completedDateIndex + 1).setValue(new Date());
    sheet.getRange(rowIndex, completedByIndex + 1).setValue(completedBy);

    // Log activity
    logSystemActivity('Task Management', `Completed task: ${taskName}`);

    return {
      success: true,
      message: 'Task marked as completed'
```

```
    };
  } catch (error) {
    logError('completeTask', error);
    return {
      success: false,
      message: 'Error completing task: ' + error.message
    };
  }
 }
};

// Expose functions to UI
function createTask(taskData) {
 return TaskService.createTask(taskData);
}

function createAsanaTask(taskData) {
 return TaskService.createAsanaTask(taskData);
}

function getTasks(options) {
 return TaskService.getTasks(options);
}

function completeTask(rowIndex, completedBy) {
 return TaskService.completeTask(rowIndex, completedBy);
}
/**
* Service for managing team-related functionality
*/
const TeamService = {
 /**
  * Gets team members
```

```
  * @return {Array} Team members
  */
 getTeamMembers: function() {
   try {
     const ss = SpreadsheetApp.getActiveSpreadsheet();
     const sheet =
ss.getSheetByName(CRISIS_SUPPORT_CONFIG.SHEETS.COUNSELOR_TRACKING);

     if (!sheet) {
       return [];
     }

     // Get all data
     const data = sheet.getDataRange().getValues();

     // Skip header row
     if (data.length <= 1) {
       return [];
     }

     // Get column indices from headers
     const headers = data[0];
     const nameIndex = headers.indexOf('Name');
     const emailIndex = headers.indexOf('Email');
     const roleIndex = headers.indexOf('Role');
     const statusIndex = headers.indexOf('Status');
     const startDateIndex = headers.indexOf('Start Date');
     const phoneIndex = headers.indexOf('Phone');

     // Check if required columns exist
     if (nameIndex === -1 || emailIndex === -1) {
       throw new Error('Required columns not found in Counselor Tracking
sheet');
```

```
    }

    // Map data to team members
    const teamMembers = [];

    for (let i = 1; i < data.length; i++) {
      const row = data[i];

      // Skip empty rows
      if (!row[nameIndex] || !row[emailIndex]) {
        continue;
      }

      teamMembers.push({
        name: row[nameIndex],
        email: row[emailIndex],
        role: roleIndex !== -1 ? row[roleIndex] : '',
        status: statusIndex !== -1 ? row[statusIndex] : 'active',
        startDate: startDateIndex !== -1 ? row[startDateIndex] : '',
        phone: phoneIndex !== -1 ? row[phoneIndex] : ''
      });
    }

    return teamMembers;
  } catch (error) {
    logError('getTeamMembers', error);
    return [];
  }
},
 /**
 * Gets team member by email
 * @param {string} email - Email of the team member
 * @return {object} Team member
```

```
  */
 getTeamMemberByEmail: function(email) {
   try {
     const teamMembers = this.getTeamMembers();
     return teamMembers.find(member => member.email === email);
   } catch (error) {
     logError('getTeamMemberByEmail', error);
     return null;
   }
 },
  /**
  * Adds a new counselor
  * @param {object} counselorData - Data for the new counselor
  * @return {object} Result of the operation
  */
 addCounselor: function(counselorData) {
   try {
     const ss = SpreadsheetApp.getActiveSpreadsheet();
     let sheet =
ss.getSheetByName(CRISIS_SUPPORT_CONFIG.SHEETS.COUNSELOR_TRACKING);

     // Create sheet if it doesn't exist
     if (!sheet) {
       sheet =
ss.insertSheet(CRISIS_SUPPORT_CONFIG.SHEETS.COUNSELOR_TRACKING);
       sheet.appendRow([
         'Name',
         'Email',
         'Role',
         'Status',
         'Start Date',
         'Phone',
         'Supervisor',
```

```
      'Certifications',
      'Notes',
      'Date Added'
    ]);
    sheet.getRange(1, 1, 1, 10)
      .setFontWeight('bold')
      .setBackground('#E0E0E0');
  }

  // Validate required fields
  if (!counselorData.name || !counselorData.email) {
    return {
      success: false,
      message: 'Name and email are required'
    };
  }

  // Check if counselor already exists
  const existingCounselors = this.getTeamMembers();
  const emailExists = existingCounselors.some(counselor =>
    counselor.email.toLowerCase() === counselorData.email.toLowerCase()
  );

  if (emailExists) {
    return {
      success: false,
      message: 'A counselor with this email already exists'
    };
  }

  // Prepare row data
  const rowData = [
    counselorData.name,
```

```
      counselorData.email,
      counselorData.role || 'Counselor',
      counselorData.status || 'active',
      counselorData.startDate || new Date(),
      counselorData.phone || '',
      counselorData.supervisor || '',
      counselorData.certifications || '',
      counselorData.notes || '',
      new Date()  // Date Added
    ];

    // Add to sheet
    sheet.appendRow(rowData);

    // Log activity
    logSystemActivity('Team Management', `Added new counselor:
${counselorData.name}`);

    return {
      success: true,
      message: 'Counselor added successfully'
    };
  } catch (error) {
    logError('addCounselor', error);
    return {
      success: false,
      message: 'Error adding counselor: ' + error.message
    };
  }
},
 /**
 * Updates a counselor's status
 * @param {object} updateData - Data for the status update
```

```
  * @return {object} Result of the operation
  */
 updateCounselorStatus: function(updateData) {
   try {
     const ss = SpreadsheetApp.getActiveSpreadsheet();
     const sheet =
ss.getSheetByName(CRISIS_SUPPORT_CONFIG.SHEETS.COUNSELOR_TRACKING);

     if (!sheet) {
       return {
         success: false,
         message: 'Counselor Tracking sheet not found'
       };
     }

     // Validate required fields
     if (!updateData.email || !updateData.status || !updateData.notes) {
       return {
         success: false,
         message: 'Email, status and notes are required'
       };
     }

     // Get all data
     const data = sheet.getDataRange().getValues();

     // Get column indices
     const headers = data[0];
     const emailIndex = headers.indexOf('Email');
     const statusIndex = headers.indexOf('Status');
     const notesIndex = headers.indexOf('Notes');
     const returnDateIndex = headers.indexOf('Return Date');
```

```
    // Check if required columns exist
    if (emailIndex === -1 || statusIndex === -1) {
      return {
        success: false,
        message: 'Required columns not found in Counselor Tracking sheet'
      };
    }

    // Find counselor
    let counselorRow = -1;

    for (let i = 1; i < data.length; i++) {
      if (data[i][emailIndex].toLowerCase() ===
updateData.email.toLowerCase()) {
        counselorRow = i + 1; // +1 because sheet rows are 1-indexed
        break;
      }
    }

    if (counselorRow === -1) {
      return {
        success: false,
        message: 'Counselor not found'
      };
    }

    // Update status
    sheet.getRange(counselorRow, statusIndex +
1).setValue(updateData.status);

    // Update return date if provided and column exists
    if (updateData.status === 'leave' && updateData.returnDate &&
returnDateIndex !== -1) {
```

```
      sheet.getRange(counselorRow, returnDateIndex +
1).setValue(updateData.returnDate);
    }

    // Update notes if provided and column exists
    if (updateData.notes && notesIndex !== -1) {
      const currentNotes = sheet.getRange(counselorRow, notesIndex +
1).getValue();
      const dateStr = new Date().toLocaleString();
      const effectiveDateStr = updateData.effectiveDate ?
        `effective ${new
Date(updateData.effectiveDate).toLocaleDateString()}` : '';
      const returnDateStr = (updateData.status === 'leave' &&
updateData.returnDate) ?
        ` with expected return on ${new
Date(updateData.returnDate).toLocaleDateString()}` : '';

      const newNotes = `${dateStr} - Status changed to ${updateData.status}
${effectiveDateStr}${returnDateStr}: ${updateData.notes}\n${currentNotes}`;

      sheet.getRange(counselorRow, notesIndex + 1).setValue(newNotes);
    }

    // Get counselor name for logging
    const nameIndex = headers.indexOf('Name');
    const counselorName = nameIndex !== -1 ? data[counselorRow -
1][nameIndex] : updateData.email;

    // Log activity
    logSystemActivity('Team Management', `Updated status for ${counselorName}
to ${updateData.status}`);

    return {
```

```
      success: true,
      message: 'Status updated successfully'
    };
  } catch (error) {
    logError('updateCounselorStatus', error);
    return {
      success: false,
      message: 'Error updating status: ' + error.message
    };
  }
},
 /**
 * Adds a coaching note for a counselor
 * @param {string} email - Email of the counselor
 * @param {object} coachingData - Coaching note data
 * @return {object} Result of the operation
 */
addCoachingNote: function(email, coachingData) {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    let sheet = ss.getSheetByName('Coaching Notes');

    // Create sheet if it doesn't exist
    if (!sheet) {
      sheet = ss.insertSheet('Coaching Notes');
      sheet.appendRow([
        'Date',
        'Counselor',
        'Email',
        'Coach',
        'Type',
        'Strengths',
        'Areas for Improvement',
```

```
      'Action Items',
      'Follow-up Date',
      'Status'
    ]);
    sheet.getRange(1, 1, 1, 10)
      .setFontWeight('bold')
      .setBackground('#E0E0E0');
  }

  // Find counselor information
  const counselor = this.getTeamMemberByEmail(email);

  if (!counselor) {
    return {
      success: false,
      message: 'Counselor not found'
    };
  }

  // Prepare row data
  const rowData = [
    new Date(),
    counselor.name,
    counselor.email,
    Session.getActiveUser().getEmail(),
    coachingData.type || 'General',
    coachingData.strengths || '',
    coachingData.areasForImprovement || '',
    coachingData.actionItems || '',
    coachingData.followUpDate || '',
    'Open'
  ];
```

```
    // Add to sheet
    sheet.appendRow(rowData);

    // Log activity
    logSystemActivity('Team Management', `Added coaching note for
${counselor.name}`);

    return {
      success: true,
      message: 'Coaching note added successfully'
    };
  } catch (error) {
    logError('addCoachingNote', error);
    return {
      success: false,
      message: 'Error adding coaching note: ' + error.message
    };
  }
},
 /**
 * Gets coaching notes for a counselor
 * @param {string} email - Email of the counselor
 * @return {Array} Coaching notes
 */
getCoachingNotes: function(email) {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    const sheet = ss.getSheetByName('Coaching Notes');

    if (!sheet) {
      return [];
    }
```

```
    // Get all data
    const data = sheet.getDataRange().getValues();

    // Skip if only header row
    if (data.length <= 1) {
      return [];
    }

    // Get column indices
    const headers = data[0];
    const dateIndex = headers.indexOf('Date');
    const emailIndex = headers.indexOf('Email');
    const coachIndex = headers.indexOf('Coach');
    const typeIndex = headers.indexOf('Type');
    const strengthsIndex = headers.indexOf('Strengths');
    const improvementIndex = headers.indexOf('Areas for Improvement');
    const actionItemsIndex = headers.indexOf('Action Items');
    const followUpIndex = headers.indexOf('Follow-up Date');
    const statusIndex = headers.indexOf('Status');

    // Filter notes for this counselor
    const notes = [];

    for (let i = 1; i < data.length; i++) {
      if (data[i][emailIndex].toLowerCase() === email.toLowerCase()) {
        notes.push({
          date: data[i][dateIndex],
          coach: data[i][coachIndex],
          type: data[i][typeIndex],
          strengths: data[i][strengthsIndex],
          areasForImprovement: data[i][improvementIndex],
          actionItems: data[i][actionItemsIndex],
          followUpDate: data[i][followUpIndex],
```

```
          status: data[i][statusIndex]
        });
      }
    }

    // Sort by date (newest first)
    notes.sort((a, b) => b.date - a.date);

    return notes;
  } catch (error) {
    logError('getCoachingNotes', error);
    return [];
  }
},
 /**
 * Gets one-on-one notes for a counselor
 * @param {string} email - Email of the counselor
 * @return {Array} One-on-one notes
 */
getOneOnOneNotes: function(email) {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    const sheet = ss.getSheetByName('1:1 Notes');

    if (!sheet) {
      return [];
    }

    // Get all data
    const data = sheet.getDataRange().getValues();

    // Skip if only header row
    if (data.length <= 1) {
```

```
      return [];
    }

    // Get column indices
    const headers = data[0];
    const dateIndex = headers.indexOf('Date');
    const counselorIndex = headers.indexOf('Counselor');
    const emailIndex = headers.indexOf('Email');
    const notesIndex = headers.indexOf('Notes');
    const actionItemsIndex = headers.indexOf('Action Items');
    const followUpIndex = headers.indexOf('Follow-up Date');

    // Filter notes for this counselor
    const notes = [];

    for (let i = 1; i < data.length; i++) {
      if (data[i][emailIndex].toLowerCase() === email.toLowerCase()) {
        notes.push({
          date: data[i][dateIndex],
          counselor: data[i][counselorIndex],
          notes: data[i][notesIndex],
          actionItems: data[i][actionItemsIndex],
          followUpDate: data[i][followUpIndex]
        });
      }
    }

    // Sort by date (newest first)
    notes.sort((a, b) => b.date - a.date);

    return notes;
  } catch (error) {
    logError('getOneOnOneNotes', error);
```

```
    return [];
  }
},
 /**
 * Adds one-on-one notes for a counselor
 * @param {string} email - Email of the counselor
 * @param {object} noteData - One-on-one note data
 * @return {object} Result of the operation
 */
addOneOnOneNote: function(email, noteData) {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    let sheet = ss.getSheetByName('1:1 Notes');

    // Create sheet if it doesn't exist
    if (!sheet) {
      sheet = ss.insertSheet('1:1 Notes');
      sheet.appendRow([
        'Date',
        'Counselor',
        'Email',
        'Manager',
        'Notes',
        'Action Items',
        'Follow-up Date',
        'Completed'
      ]);
      sheet.getRange(1, 1, 1, 8)
        .setFontWeight('bold')
        .setBackground('#E0E0E0');
    }

    // Find counselor information
```

```
    const counselor = this.getTeamMemberByEmail(email);

    if (!counselor) {
      return {
        success: false,
        message: 'Counselor not found'
      };
    }

    // Prepare row data
    const rowData = [
      new Date(),
      counselor.name,
      counselor.email,
      Session.getActiveUser().getEmail(),
      noteData.notes || '',
      noteData.actionItems || '',
      noteData.followUpDate || '',
      'No'
    ];

    // Add to sheet
    sheet.appendRow(rowData);

    // Log activity
    logSystemActivity('Team Management', `Added 1:1 note for
${counselor.name}`);

    return {
      success: true,
      message: '1:1 note added successfully'
    };
  } catch (error) {
```

```
    logError('addOneOnOneNote', error);
    return {
      success: false,
      message: 'Error adding 1:1 note: ' + error.message
    };
  }
},
 /**
 * Gets current user information
 * @return {object} User information
 */
getUserInfo: function() {
  try {
    const userEmail = Session.getActiveUser().getEmail();
    const teamMember = this.getTeamMemberByEmail(userEmail);

    if (!teamMember) {
      // Default information if not found in team members
      return {
        name: userEmail.split('@')[0],
        email: userEmail,
        role: 'User',
        status: 'online'
      };
    }

    return {
      name: teamMember.name,
      email: teamMember.email,
      role: teamMember.role,
      status: 'online'
    };
  } catch (error) {
```

```
    logError('getUserInfo', error);

    // Default fallback
    return {
      name: 'Unknown User',
      email: 'unknown@example.com',
      role: 'User',
      status: 'offline'
    };
  }
 }
};

// Expose functions to UI
function getTeamMembers() {
 return TeamService.getTeamMembers();
}

function getTeamMemberByEmail(email) {
 return TeamService.getTeamMemberByEmail(email);
}

function addCounselor(counselorData) {
 return TeamService.addCounselor(counselorData);
}

function updateCounselorStatus(updateData) {
 return TeamService.updateCounselorStatus(updateData);
}

function addCoachingNote(email, coachingData) {
 return TeamService.addCoachingNote(email, coachingData);
}
```

```
function getCoachingNotes(email) {
 return TeamService.getCoachingNotes(email);
}

function getOneOnOneNotes(email) {
 return TeamService.getOneOnOneNotes(email);
}

function addOneOnOneNote(email, noteData) {
 return TeamService.addOneOnOneNote(email, noteData);
}

function getUserInfo() {
 return TeamService.getUserInfo();
}
/**
* Service for managing time tracking functionality
*/
const TimeTrackerService = {
 /**
  * Records an activity timestamp in the Activity Breakdown sheet
  * @param {object} activityData - Data about the activity
  * @return {object} Result of the operation
  */
 recordActivityTimestamp: function(activityData) {
   try {
     const ss = SpreadsheetApp.getActiveSpreadsheet();
     let sheet = ss.getSheetByName('Activity Breakdown');

     // Create sheet if it doesn't exist
     if (!sheet) {
       sheet = ss.insertSheet('Activity Breakdown');
```

```
      sheet.appendRow([
        'Date',
        'User',
        'Activity Type',
        'Start Time',
        'End Time',
        'Duration (seconds)',
        'Duration (formatted)',
        'Notes'
      ]);
      sheet.getRange(1, 1, 1, 8)
        .setFontWeight('bold')
        .setBackground('#E0E0E0');
    }

    // Format duration
    const durationFormatted =
this.formatDuration(activityData.durationSeconds);

    // Add activity record
    sheet.appendRow([
      activityData.date,
      Session.getActiveUser().getEmail(),
      activityData.activityType,
      activityData.startTime,
      activityData.endTime,
      activityData.durationSeconds,
      durationFormatted,
      activityData.notes || ''
    ]);

    // Update daily time log
    this.updateDailyTimeLog(activityData);
```

```
    // Log the activity
    logSystemActivity('Time Tracking', `Recorded ${durationFormatted} of
${activityData.activityType} activity`);

    return {
      success: true,
      message: 'Activity recorded successfully'
    };
  } catch (error) {
    logError('recordActivityTimestamp', error);
    return {
      success: false,
      message: 'Error recording activity: ' + error.message
    };
  }
},
 /**
 * Updates the daily time log based on activity
 * @param {object} activityData - Data about the activity
 */
updateDailyTimeLog: function(activityData) {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    let sheet = ss.getSheetByName('Time Logs');

    // Create sheet if it doesn't exist
    if (!sheet) {
      sheet = ss.insertSheet('Time Logs');
      sheet.appendRow([
        'Date',
        'Primary Shift Time',
        'Admin Time',
```

```
        'Meeting Time',
        '1:1 Time',
        'Other Time',
        'Total Time',
        'Activity Notes',
        'Tasks Accomplished',
        'Summary Notes',
        'User',
        'Timestamp'
      ]);
      sheet.getRange(1, 1, 1, 12)
        .setFontWeight('bold')
        .setBackground('#E0E0E0');
    }

    // Check if there's an entry for today
    const today = activityData.date;
    const user = Session.getActiveUser().getEmail();

    // Look for existing entry for today and this user
    const data = sheet.getDataRange().getValues();
    let rowIndex = -1;

    for (let i = 1; i < data.length; i++) {
      if (data[i][0] === today && data[i][10] === user) {
        rowIndex = i + 1; // +1 because array is 0-indexed but sheet is
1-indexed
        break;
      }
    }

    // If no entry for today, create one
    if (rowIndex === -1) {
```

```
      sheet.appendRow([
        today,  // Date
        0,      // Primary Shift Time
        0,      // Admin Time
        0,      // Meeting Time
        0,      // 1:1 Time
        0,      // Other Time
        0,      // Total Time
        '',     // Activity Notes
        '',     // Tasks Accomplished
        '',     // Summary Notes
        user,   // User
        new Date() // Timestamp
      ]);
      rowIndex = sheet.getLastRow();
    }

    // Determine which column to update based on activity type
    let columnIndex;
    switch (activityData.activityType) {
      case 'Primary Shift':
        columnIndex = 2; // B column
        break;
      case 'Administrative':
        columnIndex = 3; // C column
        break;
      case 'Meetings':
        columnIndex = 4; // D column
        break;
      case '1:1 Coaching':
        columnIndex = 5; // E column
        break;
      case 'Other':
```

```
          columnIndex = 6; // F column
          break;
        default:
          throw new Error('Unknown activity type: ' +
activityData.activityType);
      }

      // Get current value
      const currentValue = sheet.getRange(rowIndex, columnIndex).getValue();

      // Add seconds to the current value
      const newValue = currentValue + activityData.durationSeconds;

      // Update the value
      sheet.getRange(rowIndex, columnIndex).setValue(newValue);

      // Update total time
      const totalSecondsRange = sheet.getRange(rowIndex, 2, 1, 5); // B-F
columns
      const totalSecondsValues = totalSecondsRange.getValues()[0];
      const totalSeconds = totalSecondsValues.reduce((sum, seconds) => sum +
(seconds || 0), 0);

      // Update total column
      sheet.getRange(rowIndex, 7).setValue(totalSeconds); // G column

      // Update timestamp
      sheet.getRange(rowIndex, 12).setValue(new Date()); // L column

      // Add to activity notes if provided
      if (activityData.notes) {
        const currentNotes = sheet.getRange(rowIndex, 8).getValue(); // H
column
```

```
      const newNotes = currentNotes ?
        `${currentNotes}\n- ${activityData.activityType}
(${this.formatDuration(activityData.durationSeconds)}): ${activityData.notes}`
:
        `- ${activityData.activityType}
(${this.formatDuration(activityData.durationSeconds)}):
${activityData.notes}`;

      sheet.getRange(rowIndex, 8).setValue(newNotes);
    }

    // Update weekly summary
    this.updateWeeklySummary();
  } catch (error) {
    logError('updateDailyTimeLog', error);
    throw error;
  }
},
 /**
 * Updates the weekly time summary
 */
updateWeeklySummary: function() {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    let timeLogs = ss.getSheetByName('Time Logs');
    let summary = ss.getSheetByName('Time Summary');

    // Skip if time logs don't exist
    if (!timeLogs) {
      return;
    }

    // Create summary sheet if it doesn't exist
```

```
  if (!summary) {
    summary = ss.insertSheet('Time Summary');
    summary.appendRow([
      'Week',
      'User',
      'Primary Shift Hours',
      'Admin Hours',
      'Meeting Hours',
      '1:1 Hours',
      'Other Hours',
      'Total Hours',
      'Tasks Count',
      'Last Updated'
    ]);
    summary.getRange(1, 1, 1, 10)
      .setFontWeight('bold')
      .setBackground('#E0E0E0');
  }

  // Get all time log data
  const timeLogData = timeLogs.getDataRange().getValues();

  // Skip if only header row
  if (timeLogData.length <= 1) {
    return;
  }

  // Group data by user and week
  const weeklyData = {};
  const user = Session.getActiveUser().getEmail();

  for (let i = 1; i < timeLogData.length; i++) {
    const row = timeLogData[i];
```

```
      if (row[10] !== user) continue; // Skip other users

      const date = new Date(row[0]); // Assuming date is in column A
      if (isNaN(date.getTime())) continue; // Skip invalid dates

      // Get week starting Monday
      const weekStart = new Date(date);
      weekStart.setDate(date.getDate() - date.getDay() + (date.getDay() === 0
? -6 : 1));
      const weekKey = weekStart.toISOString().split('T')[0]; // YYYY-MM-DD
format

      // Initialize week data if not exists
      if (!weeklyData[weekKey]) {
        weeklyData[weekKey] = {
          primaryShift: 0,
          admin: 0,
          meeting: 0,
          coaching: 0,
          other: 0,
          total: 0,
          tasksCount: 0
        };
      }

      // Add time data
      weeklyData[weekKey].primaryShift += row[1] || 0; // B column - Primary
Shift
      weeklyData[weekKey].admin += row[2] || 0; // C column - Admin
      weeklyData[weekKey].meeting += row[3] || 0; // D column - Meeting
      weeklyData[weekKey].coaching += row[4] || 0; // E column - 1:1
      weeklyData[weekKey].other += row[5] || 0; // F column - Other
      weeklyData[weekKey].total += row[6] || 0; // G column - Total
```

```
      // Count tasks by counting lines in the Tasks Accomplished column
      const tasks = row[8] ? row[8].toString().split('\n').filter(line =>
line.trim()).length : 0;
      weeklyData[weekKey].tasksCount += tasks;
    }

    // Get existing summary data
    const summaryData = summary.getDataRange().getValues();

    // Update summary sheet
    for (const weekKey in weeklyData) {
      const weekData = weeklyData[weekKey];

      // Convert seconds to hours
      const primaryShiftHours = weekData.primaryShift / 3600;
      const adminHours = weekData.admin / 3600;
      const meetingHours = weekData.meeting / 3600;
      const coachingHours = weekData.coaching / 3600;
      const otherHours = weekData.other / 3600;
      const totalHours = weekData.total / 3600;

      // Check if week exists in summary
      let rowIndex = -1;
      for (let i = 1; i < summaryData.length; i++) {
        if (summaryData[i][0] === weekKey && summaryData[i][1] === user) {
          rowIndex = i + 1; // +1 because array is 0-indexed but sheet is
1-indexed
          break;
        }
      }

      // If week exists, update it
```

```
      if (rowIndex !== -1) {
        summary.getRange(rowIndex, 3, 1, 8).setValues([[
          primaryShiftHours,
          adminHours,
          meetingHours,
          coachingHours,
          otherHours,
          totalHours,
          weekData.tasksCount,
          new Date()
        ]]);
      }
      // Otherwise, add new row
      else {
        summary.appendRow([
          weekKey,
          user,
          primaryShiftHours,
          adminHours,
          meetingHours,
          coachingHours,
          otherHours,
          totalHours,
          weekData.tasksCount,
          new Date()
        ]);
      }
    }
  } catch (error) {
    logError('updateWeeklySummary', error);
    // Don't throw, as this is a non-critical operation
  }
},
```

```
 /**
 * Gets the time log for today
 * @return {object} Today's time log data
 */
getTodaysTimeLog: function() {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    const sheet = ss.getSheetByName('Time Logs');

    // If sheet doesn't exist, return not found
    if (!sheet) {
      return { found: false };
    }

    // Get today's date in YYYY-MM-DD format
    const today = new Date().toISOString().split('T')[0];
    const user = Session.getActiveUser().getEmail();

    // Look for today's entry
    const data = sheet.getDataRange().getValues();

    for (let i = 1; i < data.length; i++) {
      const rowDate = data[i][0];
      const rowUser = data[i][10];

      // Skip if not today or not current user
      if (rowDate !== today || rowUser !== user) {
        continue;
      }

      // Found today's entry
      return {
        found: true,
```

```
        times: {
          'Primary Shift': data[i][1] || 0,
          'Administrative': data[i][2] || 0,
          'Meetings': data[i][3] || 0,
          '1:1 Coaching': data[i][4] || 0,
          'Other': data[i][5] || 0
        },
        total: data[i][6] || 0,
        notes: data[i][7] || '',
        tasks: data[i][8] || '',
        summary: data[i][9] || ''
      };
    }

    // No entry found for today
    return { found: false };
  } catch (error) {
    logError('getTodaysTimeLog', error);
    return {
      found: false,
      error: error.message
    };
  }
},
 /**
 * Gets recent activities for display in the time tracker
 * @return {Array} Recent activities
 */
getRecentActivities: function() {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    const sheet = ss.getSheetByName('Activity Breakdown');
```

```
    // If sheet doesn't exist or is empty, return empty array
    if (!sheet || sheet.getLastRow() <= 1) {
      return [];
    }

    // Get today's date in YYYY-MM-DD format
    const today = new Date().toISOString().split('T')[0];
    const user = Session.getActiveUser().getEmail();

    // Get all activity data
    const data = sheet.getDataRange().getValues();
    const activities = [];

    // Process activities (newest first)
    for (let i = data.length - 1; i > 0; i--) {
      const row = data[i];

      // Skip if not today or not current user
      if (row[0] !== today || row[1] !== user) {
        continue;
      }

      activities.push({
        activityType: row[2],
        startTime: row[3],
        endTime: row[4],
        durationSeconds: row[5],
        durationFormatted: row[6],
        notes: row[7] || ''
      });

      // Limit to 10 recent activities
      if (activities.length >= 10) {
```

```
          break;
        }
      }

      return activities;
    } catch (error) {
      logError('getRecentActivities', error);
      return [];
    }
  },
  /**
  * Formats seconds into a readable duration string (HH:MM:SS)
  * @param {number} seconds - Number of seconds
  * @return {string} Formatted duration
  */
  formatDuration: function(seconds) {
    const h = Math.floor(seconds / 3600);
    const m = Math.floor((seconds % 3600) / 60);
    const s = seconds % 60;

    return `${h.toString().padStart(2, '0')}:${m.toString().padStart(2,
'0')}:${s.toString().padStart(2, '0')}`;
  }
};

/**
* Record activity timestamp (exposed to UI)
*/
function recordActivityTimestamp(activityData) {
  return TimeTrackerService.recordActivityTimestamp(activityData);
}

/**
```

```
* Get today's time log (exposed to UI)
*/
function getTodaysTimeLog() {
 return TimeTrackerService.getTodaysTimeLog();
}

/**
* Get recent activities (exposed to UI)
*/
function getRecentActivities() {
 return TimeTrackerService.getRecentActivities();
}

/**
* Save time log (exposed to UI)
*/
function saveTimeLog(logData) {
 try {
   const ss = SpreadsheetApp.getActiveSpreadsheet();
   let sheet = ss.getSheetByName('Time Logs');

   // Create sheet if it doesn't exist
   if (!sheet) {
     sheet = ss.insertSheet('Time Logs');
     sheet.appendRow([
       'Date',
       'Primary Shift Time',
       'Admin Time',
       'Meeting Time',
       '1:1 Time',
       'Other Time',
       'Total Time',
       'Activity Notes',
```

```
      'Tasks Accomplished',
      'Summary Notes',
      'User',
      'Timestamp'
    ]);
    sheet.getRange(1, 1, 1, 12)
      .setFontWeight('bold')
      .setBackground('#E0E0E0');
  }

  // Check if there's an entry for today
  const today = new Date().toISOString().split('T')[0];
  const user = Session.getActiveUser().getEmail();

  // Look for existing entry for today and this user
  const data = sheet.getDataRange().getValues();
  let rowIndex = -1;

  for (let i = 1; i < data.length; i++) {
    if (data[i][0] === today && data[i][10] === user) {
      rowIndex = i + 1; // +1 because array is 0-indexed but sheet is
1-indexed
      break;
    }
  }

  // Calculate total time
  const totalSeconds =
    (logData.primaryShiftTime || 0) +
    (logData.adminTime || 0) +
    (logData.meetingTime || 0) +
    (logData.coachingTime || 0) +
    (logData.otherTime || 0);
```

```
// If entry exists, update it
if (rowIndex !== -1) {
  sheet.getRange(rowIndex, 2, 1, 6).setValues([[
    logData.primaryShiftTime || 0,
    logData.adminTime || 0,
    logData.meetingTime || 0,
    logData.coachingTime || 0,
    logData.otherTime || 0,
    totalSeconds
  ]]);

  // Update notes if provided
  if (logData.activityNotes) {
    sheet.getRange(rowIndex, 8).setValue(logData.activityNotes);
  }

  // Update tasks if provided
  if (logData.tasksAccomplished) {
    sheet.getRange(rowIndex, 9).setValue(logData.tasksAccomplished);
  }

  // Update summary if provided
  if (logData.summaryNotes) {
    sheet.getRange(rowIndex, 10).setValue(logData.summaryNotes);
  }

  // Update timestamp
  sheet.getRange(rowIndex, 12).setValue(new Date());
}
// Otherwise, create new entry
else {
  sheet.appendRow([
```

```
      today,
      logData.primaryShiftTime || 0,
      logData.adminTime || 0,
      logData.meetingTime || 0,
      logData.coachingTime || 0,
      logData.otherTime || 0,
      totalSeconds,
      logData.activityNotes || '',
      logData.tasksAccomplished || '',
      logData.summaryNotes || '',
      user,
      new Date()
    ]);
  }

  // Update weekly summary
  TimeTrackerService.updateWeeklySummary();

  // Log the activity
  logSystemActivity('Time Tracking', 'Saved time log');

  return {
    success: true,
    message: 'Time log saved successfully'
  };
 } catch (error) {
  logError('saveTimeLog', error);
  return {
    success: false,
    message: 'Error saving time log: ' + error.message
  };
 }
}
```

```
/**
* Service for managing UI-related functionality
*/
const UIService = {

/**
* Shows the quality review form
*/
showQualityReviewForm: function() {
 try {
   const html = HtmlService.createHtmlOutputFromFile('quality-review-form')
     .setTitle('Quality Review')
     .setWidth(900)
     .setHeight(800);

   SpreadsheetApp.getUi().showModalDialog(html, 'Quality Review');

   logSystemActivity('UI', 'Showed quality review form');
 } catch (error) {
   logError('showQualityReviewForm', error);

   // Show error message
   SpreadsheetApp.getUi().alert(
     'Error',
     'An error occurred while showing the quality review form: ' +
error.message,
     SpreadsheetApp.getUi().ButtonSet.OK
   );
 }
},

/**
* Shows the quality reports
```

```
*/
showQualityReports: function() {
 try {
   const html = HtmlService.createHtmlOutputFromFile('quality-reports')
     .setTitle('Quality Reports')
     .setWidth(900)
     .setHeight(700);

   SpreadsheetApp.getUi().showModalDialog(html, 'Quality Reports');

   logSystemActivity('UI', 'Showed quality reports');
 } catch (error) {
   logError('showQualityReports', error);

   // Show error message
   SpreadsheetApp.getUi().alert(
     'Error',
     'An error occurred while showing the quality reports: ' + error.message,
     SpreadsheetApp.getUi().ButtonSet.OK
   );
 }
},

 showSidebar: function() {
   try {
     const html = HtmlService.createHtmlOutputFromFile('sidebar')
       .setTitle('Crisis Support')
       .setWidth(300);

     SpreadsheetApp.getUi().showSidebar(html);

     logSystemActivity('UI', 'Opened sidebar');
   } catch (error) {
```

```
    logError('showSidebar', error);

    // Show error message
    SpreadsheetApp.getUi().alert(
      'Error',
      'An error occurred while showing the sidebar: ' + error.message,
      SpreadsheetApp.getUi().ButtonSet.OK
    );
  }
},
 /**
 * Shows the dashboard
 */
showDashboard: function() {
  try {
    const html = HtmlService.createHtmlOutputFromFile('dashboard')
      .setTitle('Crisis Support Dashboard')
      .setWidth(900)
      .setHeight(600);

    SpreadsheetApp.getUi().showModalDialog(html, 'Crisis Support Dashboard');

    logSystemActivity('UI', 'Opened dashboard');
  } catch (error) {
    logError('showDashboard', error);

    // Show error message
    SpreadsheetApp.getUi().alert(
      'Error',
      'An error occurred while showing the dashboard: ' + error.message,
      SpreadsheetApp.getUi().ButtonSet.OK
    );
  }
```

```
},
 /**
 * Shows the add counselor form
 */
showAddCounselorForm: function() {
  try {
    const html = HtmlService.createHtmlOutputFromFile('add-counselor-form')
      .setTitle('Add Counselor')
      .setWidth(500)
      .setHeight(600);

    SpreadsheetApp.getUi().showModalDialog(html, 'Add Counselor');

    logSystemActivity('UI', 'Opened add counselor form');
  } catch (error) {
    logError('showAddCounselorForm', error);

    // Show error message
    SpreadsheetApp.getUi().alert(
      'Error',
      'An error occurred while showing the add counselor form: ' +
error.message,
      SpreadsheetApp.getUi().ButtonSet.OK
    );
  }
},
 /**
 * Shows the update status form
 */
showUpdateStatusForm: function() {
  try {
    const html = HtmlService.createHtmlOutputFromFile('update-status-form')
      .setTitle('Update Counselor Status')
```

```
      .setWidth(500)
      .setHeight(500);

    SpreadsheetApp.getUi().showModalDialog(html, 'Update Counselor Status');

    logSystemActivity('UI', 'Opened update status form');
  } catch (error) {
    logError('showUpdateStatusForm', error);

    // Show error message
    SpreadsheetApp.getUi().alert(
      'Error',
      'An error occurred while showing the update status form: ' +
error.message,
      SpreadsheetApp.getUi().ButtonSet.OK
    );
  }
},
 /**
 * Shows the coaching form
 */
showCoachingForm: function() {
  try {
    const html = HtmlService.createHtmlOutputFromFile('coaching-form')
      .setTitle('Add Coaching Note')
      .setWidth(600)
      .setHeight(700);

    SpreadsheetApp.getUi().showModalDialog(html, 'Add Coaching Note');

    logSystemActivity('UI', 'Opened coaching form');
  } catch (error) {
    logError('showCoachingForm', error);
```

```
    // Show error message
    SpreadsheetApp.getUi().alert(
      'Error',
      'An error occurred while showing the coaching form: ' + error.message,
      SpreadsheetApp.getUi().ButtonSet.OK
    );
  }
},
 /**
 * Shows the one-on-one notes
 */
showOneOnOneNotes: function() {
  try {
    const html = HtmlService.createHtmlOutputFromFile('one-on-one-notes')
      .setTitle('Counselor 1:1 Notes')
      .setWidth(700)
      .setHeight(700);

    SpreadsheetApp.getUi().showModalDialog(html, 'Counselor 1:1 Notes');

    logSystemActivity('UI', 'Opened 1:1 notes');
  } catch (error) {
    logError('showOneOnOneNotes', error);

    // Show error message
    SpreadsheetApp.getUi().alert(
      'Error',
      'An error occurred while showing the 1:1 notes: ' + error.message,
      SpreadsheetApp.getUi().ButtonSet.OK
    );
  }
},
```

```
 /**
 * Shows the metrics form
 */
showMetricsForm: function() {
  try {
    const html = HtmlService.createHtmlOutputFromFile('metrics-form')
      .setTitle('Enter Daily Metrics')
      .setWidth(600)
      .setHeight(650);

    SpreadsheetApp.getUi().showModalDialog(html, 'Enter Daily Metrics');

    logSystemActivity('UI', 'Opened metrics form');
  } catch (error) {
    logError('showMetricsForm', error);

    // Show error message
    SpreadsheetApp.getUi().alert(
      'Error',
      'An error occurred while showing the metrics form: ' + error.message,
      SpreadsheetApp.getUi().ButtonSet.OK
    );
  }
},
 /**
 * Shows the metrics report
 */
showMetricsReport: function() {
  try {
    const html = HtmlService.createHtmlOutputFromFile('metrics-report')
      .setTitle('Metrics Report')
      .setWidth(800)
      .setHeight(700);
```

```
    SpreadsheetApp.getUi().showModalDialog(html, 'Metrics Report');

    logSystemActivity('UI', 'Opened metrics report');
  } catch (error) {
    logError('showMetricsReport', error);

    // Show error message
    SpreadsheetApp.getUi().alert(
      'Error',
      'An error occurred while showing the metrics report: ' + error.message,
      SpreadsheetApp.getUi().ButtonSet.OK
    );
  }
},
 /**
 * Shows the alert form
 */
showAlertForm: function() {
  try {
    const html = HtmlService.createHtmlOutputFromFile('alert-form')
      .setTitle('Create Alert')
      .setWidth(550)
      .setHeight(600);

    SpreadsheetApp.getUi().showModalDialog(html, 'Create Alert');

    logSystemActivity('UI', 'Opened alert form');
  } catch (error) {
    logError('showAlertForm', error);

    // Show error message
    SpreadsheetApp.getUi().alert(
```

```
      'Error',
      'An error occurred while showing the alert form: ' + error.message,
      SpreadsheetApp.getUi().ButtonSet.OK
    );
  }
},
 /**
 * Shows the active alerts
 */
showActiveAlerts: function() {
  try {
    const html = HtmlService.createHtmlOutputFromFile('active-alerts')
      .setTitle('Active Alerts')
      .setWidth(700)
      .setHeight(600);

    SpreadsheetApp.getUi().showModalDialog(html, 'Active Alerts');

    logSystemActivity('UI', 'Opened active alerts');
  } catch (error) {
    logError('showActiveAlerts', error);

    // Show error message
    SpreadsheetApp.getUi().alert(
      'Error',
      'An error occurred while showing active alerts: ' + error.message,
      SpreadsheetApp.getUi().ButtonSet.OK
    );
  }
},
 /**
 * Shows the task form
 */
```

```
showTaskForm: function() {
  try {
    const html = HtmlService.createHtmlOutputFromFile('task-form')
      .setTitle('Create Task')
      .setWidth(550)
      .setHeight(600);

    SpreadsheetApp.getUi().showModalDialog(html, 'Create Task');

    logSystemActivity('UI', 'Opened task form');
  } catch (error) {
    logError('showTaskForm', error);

    // Show error message
    SpreadsheetApp.getUi().alert(
      'Error',
      'An error occurred while showing the task form: ' + error.message,
      SpreadsheetApp.getUi().ButtonSet.OK
    );
  }
},
 /**
 * Shows the Asana task form
 */
showAsanaTaskForm: function() {
  try {
    const html = HtmlService.createHtmlOutputFromFile('asana-task-form')
      .setTitle('Create Asana Task')
      .setWidth(550)
      .setHeight(650);

    SpreadsheetApp.getUi().showModalDialog(html, 'Create Asana Task');
```

```
    logSystemActivity('UI', 'Opened Asana task form');
  } catch (error) {
    logError('showAsanaTaskForm', error);

    // Show error message
    SpreadsheetApp.getUi().alert(
      'Error',
      'An error occurred while showing the Asana task form: ' +
error.message,
      SpreadsheetApp.getUi().ButtonSet.OK
    );
  }
},
 /**
 * Shows the schedule manager
 */
showScheduleManager: function() {
  try {
    const html = HtmlService.createHtmlOutputFromFile('schedule-manager')
      .setTitle('Manage Schedule')
      .setWidth(800)
      .setHeight(700);

    SpreadsheetApp.getUi().showModalDialog(html, 'Manage Schedule');

    logSystemActivity('UI', 'Opened schedule manager');
  } catch (error) {
    logError('showScheduleManager', error);

    // Show error message
    SpreadsheetApp.getUi().alert(
      'Error',
```

```
      'An error occurred while showing the schedule manager: ' +
error.message,
      SpreadsheetApp.getUi().ButtonSet.OK
    );
  }
},
 /**
 * Shows the initialize week form
 */
showInitializeWeekForm: function() {
  try {
    const html = HtmlService.createHtmlOutputFromFile('initialize-week-form')
      .setTitle('Initialize Week')
      .setWidth(500)
      .setHeight(400);

    SpreadsheetApp.getUi().showModalDialog(html, 'Initialize Week');

    logSystemActivity('UI', 'Opened initialize week form');
  } catch (error) {
    logError('showInitializeWeekForm', error);

    // Show error message
    SpreadsheetApp.getUi().alert(
      'Error',
      'An error occurred while showing the initialize week form: ' +
error.message,
      SpreadsheetApp.getUi().ButtonSet.OK
    );
  }
},
 /**
 * Shows the shift initialization form
```

```
 */
showShiftInitialization: function() {
  try {
    const html = HtmlService.createHtmlOutputFromFile('shift-initialization')
      .setTitle('Initialize Shift')
      .setWidth(550)
      .setHeight(500);

    SpreadsheetApp.getUi().showModalDialog(html, 'Initialize Shift');

    logSystemActivity('UI', 'Opened shift initialization');
  } catch (error) {
    logError('showShiftInitialization', error);

    // Show error message
    SpreadsheetApp.getUi().alert(
      'Error',
      'An error occurred while showing the shift initialization: ' +
error.message,
      SpreadsheetApp.getUi().ButtonSet.OK
    );
  }
},
 /**
 * Shows the time tracker
 */
showTimeTracker: function() {
  try {
    const html = HtmlService.createHtmlOutputFromFile('time-tracker')
      .setTitle('Time Tracker')
      .setWidth(600)
      .setHeight(500);
```

```
    SpreadsheetApp.getUi().showModalDialog(html, 'Time Tracker');

    logSystemActivity('UI', 'Opened time tracker');
  } catch (error) {
    logError('showTimeTracker', error);

    // Show error message
    SpreadsheetApp.getUi().alert(
      'Error',
      'An error occurred while showing the time tracker: ' + error.message,
      SpreadsheetApp.getUi().ButtonSet.OK
    );
  }
},
 /**
 * Shows the crisis protocols
 */
showCrisisProtocols: function() {
  try {
    const html = HtmlService.createHtmlOutputFromFile('crisis-protocols')
      .setTitle('Crisis Protocols')
      .setWidth(700)
      .setHeight(700);

    SpreadsheetApp.getUi().showModalDialog(html, 'Crisis Protocols');

    logSystemActivity('UI', 'Opened crisis protocols');
  } catch (error) {
    logError('showCrisisProtocols', error);

    // Show error message
    SpreadsheetApp.getUi().alert(
      'Error',
```

```
      'An error occurred while showing the crisis protocols: ' +
error.message,
      SpreadsheetApp.getUi().ButtonSet.OK
    );
  }
},
 /**
 * Shows the help resources
 */
showHelp: function() {
  try {
    const html = HtmlService.createHtmlOutputFromFile('help-resources')
      .setTitle('Help Resources')
      .setWidth(700)
      .setHeight(600);

    SpreadsheetApp.getUi().showModalDialog(html, 'Help Resources');

    logSystemActivity('UI', 'Opened help resources');
  } catch (error) {
    logError('showHelp', error);

    // Show error message
    SpreadsheetApp.getUi().alert(
      'Error',
      'An error occurred while showing the help resources: ' + error.message,
      SpreadsheetApp.getUi().ButtonSet.OK
    );
  }
},
 /**
 * Shows the manager one-on-ones
 */
```

```
 showManagerOneOnOnes: function() {
   try {
     const html = HtmlService.createHtmlOutputFromFile('manager-one-on-ones')
       .setTitle('Manager 1:1 Documentation')
       .setWidth(700)
       .setHeight(700);

     SpreadsheetApp.getUi().showModalDialog(html, 'Manager 1:1
Documentation');

     logSystemActivity('UI', 'Opened manager 1:1 documentation');
   } catch (error) {
     logError('showManagerOneOnOnes', error);

     // Show error message
     SpreadsheetApp.getUi().alert(
       'Error',
       'An error occurred while showing the manager 1:1 documentation: ' +
error.message,
       SpreadsheetApp.getUi().ButtonSet.OK
     );
   }
 },
  /**
  * Shows the settings
  */
 showSettings: function() {
   try {
     const html = HtmlService.createHtmlOutputFromFile('settings')
       .setTitle('Settings')
       .setWidth(600)
       .setHeight(500);
```

```
    SpreadsheetApp.getUi().showModalDialog(html, 'Settings');

    logSystemActivity('UI', 'Opened settings');
  } catch (error) {
    logError('showSettings', error);

    // Show error message
    SpreadsheetApp.getUi().alert(
      'Error',
      'An error occurred while showing the settings: ' + error.message,
      SpreadsheetApp.getUi().ButtonSet.OK
    );
  }
 }
};

// Expose UI functions
function showSidebar() {
 UIService.showSidebar();
}

function showDashboard() {
 UIService.showDashboard();
}

function showAddCounselorForm() {
 UIService.showAddCounselorForm();
}

function showUpdateStatusForm() {
 UIService.showUpdateStatusForm();
}
```

```
function showCoachingForm() {
 UIService.showCoachingForm();
}

function showOneOnOneNotes() {
 UIService.showOneOnOneNotes();
}

function showMetricsForm() {
 UIService.showMetricsForm();
}

function showMetricsReport() {
 UIService.showMetricsReport();
}

function showAlertForm() {
 UIService.showAlertForm();
}

function showActiveAlerts() {
 UIService.showActiveAlerts();
}

function showTaskForm() {
 UIService.showTaskForm();
}

function showAsanaTaskForm() {
 UIService.showAsanaTaskForm();
}

function showScheduleManager() {
```

```
 UIService.showScheduleManager();
}

function showInitializeWeekForm() {
 UIService.showInitializeWeekForm();
}

function showShiftInitialization() {
 UIService.showShiftInitialization();
}

function showTimeTracker() {
 UIService.showTimeTracker();
}

function showCrisisProtocols() {
 UIService.showCrisisProtocols();
}

function showHelp() {
 UIService.showHelp();
}

function showManagerOneOnOnes() {
 UIService.showManagerOneOnOnes();
}

function showSettings() {
 UIService.showSettings();
}
function showQualityReviewForm() {
 UIService.showQualityReviewForm();
}
```

```
function showQualityReports() {
 UIService.showQualityReports();
}
/**
* Interactive Validation Framework for Crisis Support System
*
* Provides comprehensive sheet structure and data validation
* with interactive resolution capabilities.
*/
const ValidationSystem = {
 // Store validation state
 validationState: {
   lastRunTimestamp: null,
   validationResults: {},
   issuesFound: 0,
   issuesResolved: 0,
   inProgress: false
 },
  // Validation levels
 LEVELS: {
   CRITICAL: 'critical',
   WARNING: 'warning',
   INFO: 'info'
 },
  // Categories of validation
 CATEGORIES: {
   SHEET_STRUCTURE: 'sheet_structure',
   DATA_INTEGRITY: 'data_integrity',
   CONFIGURATION: 'configuration',
   ACCESS_PERMISSIONS: 'permissions',
   FORMULA_HEALTH: 'formulas'
 },
```

```
  /**
  * Run all validation checks and return results
  * @param {boolean} interactive - Whether to show interactive UI for issues
  * @param {boolean} autoFix - Whether to automatically fix non-critical
issues
  * @return {object} Validation results
  */
 runValidation: function(interactive = true, autoFix = false) {
   try {
     // Initialize validation state
     this.validationState = {
       lastRunTimestamp: new Date(),
       validationResults: {},
       issuesFound: 0,
       issuesResolved: 0,
       inProgress: true
     };

     // Show progress indicator if interactive
     if (interactive) {
       this.showProgressIndicator('Validating system structure...');
     }

     // Run validation checks in sequence
     this.validateSheetStructure(interactive, autoFix);
     this.validateDataIntegrity(interactive, autoFix);
     this.validateConfiguration(interactive, autoFix);
     this.validatePermissions(interactive, autoFix);
     this.validateFormulas(interactive, autoFix);

     // Summarize results
     const summary = this.summarizeValidationResults();
```

```
    // Show results if interactive
    if (interactive) {
      this.hideProgressIndicator();
      this.displayValidationResults(summary);
    }

    // Log the validation run
    this.logValidationRun(summary);

    // Update validation state
    this.validationState.inProgress = false;
    return summary;
  } catch (e) {
    console.error('Error in validation system:', e);

    // Clean up UI if interactive
    if (interactive) {
      this.hideProgressIndicator();
      this.showValidationError(e);
    }

    // Update validation state
    this.validationState.inProgress = false;
    return {
      success: false,
      error: e.message,
      timestamp: new Date()
    };
  }
},
 /**
 * Validates sheet structure against expected configuration
 */
```

```
validateSheetStructure: function(interactive, autoFix) {
  // Initialize results object for this category
  this.validationState.validationResults[this.CATEGORIES.SHEET_STRUCTURE] = {
    checks: [],
    issues: 0,
    fixed: 0
  };

  // Update progress indicator if interactive
  if (interactive) {
    this.updateProgressIndicator('Checking sheet structure...', 20);
  }

  // Get all required sheets from configuration
  const requiredSheets = this.getRequiredSheets();
  const ss = SpreadsheetApp.getActiveSpreadsheet();
  const existingSheets = ss.getSheets().map(sheet => sheet.getName());

  // Check each required sheet
  requiredSheets.forEach(sheetConfig => {
    const checkResult = {
      name: `Sheet: ${sheetConfig.name}`,
      passed: existingSheets.includes(sheetConfig.name),
      level: sheetConfig.critical ? this.LEVELS.CRITICAL :
this.LEVELS.WARNING,
      message: existingSheets.includes(sheetConfig.name)
        ? `Sheet "${sheetConfig.name}" exists`
        : `Required sheet "${sheetConfig.name}" is missing`,
      fixable: true,
      fixed: false,
      fixMessage: `Create missing sheet "${sheetConfig.name}" with proper
headers`
    };
```

```
    // If sheet doesn't exist and we should fix it
    if (!checkResult.passed && (autoFix || (interactive &&
this.shouldFixInteractively(checkResult)))) {
      try {
        // Create the missing sheet
        const newSheet = ss.insertSheet(sheetConfig.name);

        // Add headers if specified
        if (sheetConfig.headers && sheetConfig.headers.length > 0) {
          newSheet.appendRow(sheetConfig.headers);
          // Format header row
          newSheet.getRange(1, 1, 1, sheetConfig.headers.length)
            .setFontWeight('bold')
            .setBackground('#E0E0E0');
        }

        // Update check result
        checkResult.fixed = true;
        checkResult.message = `Created missing sheet "${sheetConfig.name}"`;

this.validationState.validationResults[this.CATEGORIES.SHEET_STRUCTURE].fixed+
+;
        this.validationState.issuesResolved++;
      } catch (e) {
        checkResult.fixable = false;
        checkResult.message += ` (Error during fix: ${e.message})`;
      }
    }

    // Add the check result
```

```
this.validationState.validationResults[this.CATEGORIES.SHEET_STRUCTURE].checks
.push(checkResult);

    // Increment issues count if not passed
    if (!checkResult.passed) {

this.validationState.validationResults[this.CATEGORIES.SHEET_STRUCTURE].issues
++;
      this.validationState.issuesFound++;
    }
  });

  // Validate column headers for each existing required sheet
  const existingRequiredSheets = requiredSheets
    .filter(sheetConfig => existingSheets.includes(sheetConfig.name));

  existingRequiredSheets.forEach(sheetConfig => {
    // Skip if no headers defined
    if (!sheetConfig.headers || sheetConfig.headers.length === 0) {
      return;
    }

    // Get the sheet
    const sheet = ss.getSheetByName(sheetConfig.name);

    // Get existing headers
    let existingHeaders = [];
    if (sheet.getLastRow() > 0) {
      existingHeaders = sheet.getRange(1, 1, 1,
sheet.getLastColumn()).getValues()[0];
    }
```

```
    // Compare headers
    const checkResult = {
      name: `Headers: ${sheetConfig.name}`,
      passed: this.arraysMatch(existingHeaders, sheetConfig.headers),
      level: sheetConfig.critical ? this.LEVELS.CRITICAL :
this.LEVELS.WARNING,
      message: this.arraysMatch(existingHeaders, sheetConfig.headers)
        ? `Headers for "${sheetConfig.name}" match expected configuration`
        : `Headers for "${sheetConfig.name}" do not match expected
configuration`,
      fixable: true,
      fixed: false,
      fixMessage: `Update headers for "${sheetConfig.name}" to match
configuration`
    };

    // If headers don't match and we should fix it
    if (!checkResult.passed && (autoFix || (interactive &&
this.shouldFixInteractively(checkResult)))) {
      try {
        // Clear existing headers if any
        if (sheet.getLastRow() > 0) {
          sheet.getRange(1, 1, 1, sheet.getLastColumn()).clearContent();
        }

        // Add the correct headers
        sheet.getRange(1, 1, 1, sheetConfig.headers.length)
          .setValues([sheetConfig.headers])
          .setFontWeight('bold')
          .setBackground('#E0E0E0');

        // Update check result
        checkResult.fixed = true;
```

```
          checkResult.message = `Updated headers for "${sheetConfig.name}" to
match configuration`;

this.validationState.validationResults[this.CATEGORIES.SHEET_STRUCTURE].fixed+
+;
          this.validationState.issuesResolved++;
        } catch (e) {
          checkResult.fixable = false;
          checkResult.message += ` (Error during fix: ${e.message})`;
        }
      }

      // Add the check result

this.validationState.validationResults[this.CATEGORIES.SHEET_STRUCTURE].checks
.push(checkResult);

      // Increment issues count if not passed
      if (!checkResult.passed) {

this.validationState.validationResults[this.CATEGORIES.SHEET_STRUCTURE].issues
++;
        this.validationState.issuesFound++;
      }
    });
  },
  /**
  * Validates data integrity across sheets
  */
  validateDataIntegrity: function(interactive, autoFix) {
    // Initialize results object for this category
    this.validationState.validationResults[this.CATEGORIES.DATA_INTEGRITY] = {
      checks: [],
```

```
    issues: 0,
    fixed: 0
  };

  // Update progress indicator if interactive
  if (interactive) {
    this.updateProgressIndicator('Checking data integrity...', 40);
  }

  // Basic implementation - expand as needed
  this.validateCounselorData(interactive, autoFix);
  this.validateMetricsData(interactive, autoFix);
},
 /**
 * Validates specific counselor data integrity
 */
validateCounselorData: function(interactive, autoFix) {
  const ss = SpreadsheetApp.getActiveSpreadsheet();
  const sheet =
ss.getSheetByName(CRISIS_SUPPORT_CONFIG.SHEETS.COUNSELOR_TRACKING);

  // Skip if sheet doesn't exist
  if (!sheet) {
    return;
  }

  // Get data range
  const data = sheet.getDataRange().getValues();
  if (data.length <= 1) {
    // Only header row, no data to validate
    return;
  }
```

```
  // Validate email format for all counselors
  const headers = data[0];
  const emailIndex = headers.indexOf('Email');

  // Skip if email column doesn't exist
  if (emailIndex === -1) {
    return;
  }

  // Check each row for valid email
  let invalidEmailRows = [];
  for (let i = 1; i < data.length; i++) {
    const email = data[i][emailIndex];
    if (email && !this.isValidEmail(email)) {
      invalidEmailRows.push(i + 1); // +1 for sheet row index
    }
  }

  // Create check result
  const checkResult = {
    name: 'Counselor Email Format',
    passed: invalidEmailRows.length === 0,
    level: this.LEVELS.WARNING,
    message: invalidEmailRows.length === 0
      ? 'All counselor emails have valid format'
      : `Found ${invalidEmailRows.length} counselor(s) with invalid email
format`,
    fixable: false, // Manual fix required
    fixed: false,
    fixMessage: 'Review and correct invalid email addresses',
    details: `Invalid emails found in rows: ${invalidEmailRows.join(', ')}`
  };
```

```
    // Add the check result
    this.validationState.validationResults[this.CATEGORIES.DATA_INTEGRITY].checks.push(checkResult);

    // Increment issues count if not passed
    if (!checkResult.passed) {
      this.validationState.validationResults[this.CATEGORIES.DATA_INTEGRITY].issues++;
      this.validationState.issuesFound++;
    }
  },

  /**
   * Validates metrics data consistency
   */
  validateMetricsData: function(interactive, autoFix) {
    // Basic implementation - expand as needed
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    const sheet = ss.getSheetByName(CRISIS_SUPPORT_CONFIG.SHEETS.CALL_METRICS);

    // Skip if sheet doesn't exist
    if (!sheet) {
      return;
    }

    // Additional validation logic would go here
  },

  /**
   * Validates system configuration
   */
  validateConfiguration: function(interactive, autoFix) {
    // Initialize results object for this category
```

```
  this.validationState.validationResults[this.CATEGORIES.CONFIGURATION] = {
    checks: [],
    issues: 0,
    fixed: 0
  };

  // Update progress indicator if interactive
  if (interactive) {
    this.updateProgressIndicator('Checking system configuration...', 60);
  }

  // Basic check for script properties
  const scriptProperties =
PropertiesService.getScriptProperties().getProperties();
  const requiredProperties = [
    'alertEmailRecipients',
    CRISIS_SUPPORT_CONFIG.ASANA.API_KEY_PROPERTY,
    CRISIS_SUPPORT_CONFIG.ASANA.WORKSPACE_GID_PROPERTY,
    CRISIS_SUPPORT_CONFIG.ASANA.PROJECT_GID_PROPERTY,
    'systemInitialized',
    'initializationDate'
  ];

  const missingProperties = requiredProperties.filter(prop =>
!scriptProperties[prop]);

  // Create check result
  const checkResult = {
    name: 'Script Properties Configuration',
    passed: missingProperties.length === 0,
    level: this.LEVELS.WARNING,
    message: missingProperties.length === 0
      ? 'All required script properties are configured'
```

```
        : `Missing ${missingProperties.length} required script properties`,
      fixable: false, // Complex fix requiring user input
      fixed: false,
      fixMessage: 'Configure missing script properties in Settings',
      details: `Missing properties: ${missingProperties.join(', ')}`
    };

    // Add the check result

this.validationState.validationResults[this.CATEGORIES.CONFIGURATION].checks.p
ush(checkResult);

    // Increment issues count if not passed
    if (!checkResult.passed) {

this.validationState.validationResults[this.CATEGORIES.CONFIGURATION].issues++
;
      this.validationState.issuesFound++;
    }
  },
  /**
  * Validates access permissions
  */
  validatePermissions: function(interactive, autoFix) {
    // Initialize results object for this category
    this.validationState.validationResults[this.CATEGORIES.ACCESS_PERMISSIONS]
= {
      checks: [],
      issues: 0,
      fixed: 0
    };

    // Update progress indicator if interactive
```

```
  if (interactive) {
    this.updateProgressIndicator('Checking access permissions...', 80);
  }

  // Basic implementation - would be expanded as needed
},
 /**
 * Validates formulas in the spreadsheet
 */
validateFormulas: function(interactive, autoFix) {
  // Initialize results object for this category
  this.validationState.validationResults[this.CATEGORIES.FORMULA_HEALTH] = {
    checks: [],
    issues: 0,
    fixed: 0
  };

  // Update progress indicator if interactive
  if (interactive) {
    this.updateProgressIndicator('Checking formulas...', 90);
  }

  // Basic implementation - would be expanded as needed
},
 /**
 * Summarizes the validation results
 * @return {object} Summary of validation results
 */
summarizeValidationResults: function() {
  const categories = Object.keys(this.validationState.validationResults);
  let totalChecks = 0;
  let totalIssues = 0;
  let criticalIssues = 0;
```

```
  let warningIssues = 0;
  let infoIssues = 0;
  let fixedIssues = 0;

  // Count issues by category and level
  categories.forEach(category => {
    const categoryData = this.validationState.validationResults[category];
    totalChecks += categoryData.checks.length;
    totalIssues += categoryData.issues;
    fixedIssues += categoryData.fixed;

    // Count by level
    categoryData.checks.forEach(check => {
      if (!check.passed) {
        if (check.level === this.LEVELS.CRITICAL) criticalIssues++;
        else if (check.level === this.LEVELS.WARNING) warningIssues++;
        else if (check.level === this.LEVELS.INFO) infoIssues++;
      }
    });
  });

  return {
    timestamp: this.validationState.lastRunTimestamp,
    totalChecks: totalChecks,
    issuesFound: totalIssues,
    criticalIssues: criticalIssues,
    warningIssues: warningIssues,
    infoIssues: infoIssues,
    issuesFixed: fixedIssues,
    categories: categories.map(category => ({
      name: category,
      checksRun:
this.validationState.validationResults[category].checks.length,
```

```
      issuesFound: this.validationState.validationResults[category].issues,
      issuesFixed: this.validationState.validationResults[category].fixed
    })),
    passingRate: totalChecks > 0 ? ((totalChecks - totalIssues) / totalChecks
* 100).toFixed(1) + '%' : '100%',
    success: totalIssues === 0 || totalIssues === fixedIssues
  };
},
 /**
 * Shows an interactive dialog to fix an issue
 * @param {object} issue - The issue to fix
 * @return {boolean} Whether the user chose to fix the issue
 */
shouldFixInteractively: function(issue) {
  const ui = SpreadsheetApp.getUi();

  // Create message based on severity
  let icon = '⚠';
  if (issue.level === this.LEVELS.CRITICAL) icon = '🔴';
  else if (issue.level === this.LEVELS.INFO) icon = 'ℹ';

  const message = `${icon} ${issue.level.toUpperCase()}:
${issue.message}\n\n` +
    `The system can automatically fix this issue:\n${issue.fixMessage}\n\n` +
    `Would you like to fix this issue now?`;

  const response = ui.alert(
    'System Validation',
    message,
    ui.ButtonSet.YES_NO
  );

  return response === ui.Button.YES;
```

```
},
 /**
 * Shows the validation progress indicator
 * @param {string} message - The message to display
 */
showProgressIndicator: function(message) {
  // Show a modal dialog with progress bar
  const html = HtmlService.createHtmlOutput(`
    <!DOCTYPE html>
    <html>
    <head>
      <base target="_top">
      <style>
        body {
          font-family: 'Roboto', Arial, sans-serif;
          padding: 20px;
          text-align: center;
        }
        .progress-container {
          width: 100%;
          margin: 20px 0;
        }
        .progress-bar {
          width: 100%;
          background-color: #e0e0e0;
          border-radius: 4px;
          overflow: hidden;
        }
        .progress-fill {
          height: 20px;
          width: 0%;
          background: linear-gradient(135deg, #FF786E, #FBCBBE);
          transition: width 0.3s ease;
```

```
        }
        .message {
          margin-top: 10px;
          min-height: 20px;
        }
      </style>
    </head>
    <body>
      <h2>System Validation</h2>
      <p>Please wait while the system is being validated...</p>
      <div class="progress-container">
        <div class="progress-bar">
          <div id="progress-fill" class="progress-fill"></div>
        </div>
        <div id="message" class="message">${message}</div>
      </div>
      <script>
        // Initialize progress
        document.getElementById('progress-fill').style.width = '0%';

        // Function to update progress
        function updateProgress(percent, message) {
          document.getElementById('progress-fill').style.width = percent +
'%';
          if (message) {
            document.getElementById('message').innerText = message;
          }
        }

        // Listen for progress updates
        google.script.run
          .withSuccessHandler(function(result) {
            if (result && result.close) {
```

```
              google.script.host.close();
            } else if (result) {
              updateProgress(result.percent, result.message);
            }
          })
          .withFailureHandler(function(error) {
            document.getElementById('message').innerText = 'Error: ' +
error.message;
          })
          .listenForProgressUpdates();
      </script>
    </body>
    </html>
  `)
  .setWidth(400)
  .setHeight(200);

  SpreadsheetApp.getUi().showModelessDialog(html, 'System Validation');
},
 /**
 * Updates the progress indicator
 * @param {string} message - The message to display
 * @param {number} percent - The progress percentage
 */
updateProgressIndicator: function(message, percent) {
  // Update a global variable that's checked by a function called by the
progress dialog
  PropertiesService.getScriptProperties().setProperty('validationProgress',
    JSON.stringify({
      message: message,
      percent: percent
    })
  );
```

```
},
 /**
 * Hides the progress indicator
 */
hideProgressIndicator: function() {
  // Set a flag to close the dialog
  PropertiesService.getScriptProperties().setProperty('validationProgress',
    JSON.stringify({
      close: true
    })
  );
},
 /**
 * Function called by the progress dialog to get updates
 */
listenForProgressUpdates: function() {
  const progressData =
PropertiesService.getScriptProperties().getProperty('validationProgress');
  if (progressData) {
    return JSON.parse(progressData);
  }
  return null;
},
 /**
 * Displays the validation results to the user
 * @param {object} summary - The validation summary
 */
displayValidationResults: function(summary) {
  // Create HTML for displaying results
  let htmlContent = `
    <!DOCTYPE html>
    <html>
    <head>
```

```
    <base target="_top">
    <style>
      body {
        font-family: 'Roboto', Arial, sans-serif;
        line-height: 1.6;
        color: #333;
        max-width: 800px;
        margin: 0 auto;
        padding: 20px;
      }
      h1, h2, h3 {
        color: #001A4E;
        font-family: 'Poppins', sans-serif;
      }
      .summary-card {
        background: white;
        border-radius: 12px;
        box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
        padding: 20px;
        margin-bottom: 20px;
        position: relative;
        overflow: hidden;
      }
      .summary-card::before {
        content: '';
        position: absolute;
        left: 0;
        top: 0;
        height: 100%;
        width: 4px;
        background: #FF786E;
      }
      .status-indicator {
```

```
      display: inline-block;
      padding: 4px 12px;
      border-radius: 50px;
      font-weight: 500;
      color: white;
      margin-bottom: 10px;
    }
    .status-success {
      background: #137F6A;
    }
    .status-warning {
      background: #FFAD8D;
    }
    .status-error {
      background: #F54AC;
    }
    .metrics-grid {
      display: grid;
      grid-template-columns: repeat(auto-fill, minmax(150px, 1fr));
      gap: 15px;
      margin: 20px 0;
    }
    .metric {
      background: #f5f5f5;
      padding: 10px;
      border-radius: 8px;
      text-align: center;
    }
    .metric-value {
      font-size: 24px;
      font-weight: 600;
      margin: 5px 0;
    }
```

```
        .metric-label {
          font-size: 14px;
          color: #666;
        }
        .category-card {
          background: white;
          border-radius: 8px;
          box-shadow: 0 2px 4px rgba(0, 0, 0, 0.05);
          margin-bottom: 15px;
          overflow: hidden;
        }
        .category-header {
          padding: 10px 15px;
          background: #f5f5f5;
          font-weight: 500;
          display: flex;
          justify-content: space-between;
          align-items: center;
          cursor: pointer;
        }
        .category-content {
          padding: 15px;
          display: none;
        }
        .issue-item {
          margin-bottom: 10px;
          padding-bottom: 10px;
          border-bottom: 1px solid #eee;
        }
        .issue-title {
          font-weight: 500;
          margin-bottom: 5px;
          display: flex;
```

```
    align-items: center;
  }
  .issue-badge {
    display: inline-block;
    padding: 2px 8px;
    border-radius: 50px;
    font-size: 12px;
    margin-right: 8px;
    color: white;
  }
  .badge-critical {
    background: #F54AC;
  }
  .badge-warning {
    background: #FFAD8D;
  }
  .badge-info {
    background: #4F52DE;
  }
  .badge-fixed {
    background: #137F6A;
  }
  .issue-message {
    margin: 5px 0;
  }
  .issue-details {
    background: #f5f5f5;
    padding: 10px;
    border-radius: 4px;
    font-family: monospace;
    font-size: 12px;
    white-space: pre-wrap;
    margin-top: 5px;
```

```
          display: none;
        }
        .btn {
          background: linear-gradient(135deg, #FF786E, #FBCBBE);
          color: white;
          border: none;
          padding: 8px 16px;
          border-radius: 4px;
          cursor: pointer;
          font-weight: 500;
          margin-top: 20px;
          transition: all 0.2s ease;
        }
        .btn:hover {
          transform: translateY(-2px);
          box-shadow: 0 4px 8px rgba(255, 120, 110, 0.3);
        }
      </style>
    </head>
    <body>
      <div class="summary-card">
        <h1>System Validation Results</h1>
        ${summary.success ?
          `<div class="status-indicator status-success">✓ All checks passed
or issues fixed</div>` :
          summary.criticalIssues > 0 ?
            `<div class="status-indicator status-error">✗ Critical issues
found</div>` :
            `<div class="status-indicator status-warning">⚠ Issues
found</div>`
        }
        <p>Validation completed on ${summary.timestamp.toLocaleString()}</p>
```

```
        <div class="metrics-grid">
          <div class="metric">
            <div class="metric-label">Checks Run</div>
            <div class="metric-value">${summary.totalChecks}</div>
          </div>
          <div class="metric">
            <div class="metric-label">Passing Rate</div>
            <div class="metric-value">${summary.passingRate}</div>
          </div>
          <div class="metric">
            <div class="metric-label">Issues Found</div>
            <div class="metric-value">${summary.issuesFound}</div>
          </div>
          <div class="metric">
            <div class="metric-label">Issues Fixed</div>
            <div class="metric-value">${summary.issuesFixed}</div>
          </div>
        </div>
      </div>
    `;

    // Add category sections
    summary.categories.forEach(category => {
      const categoryResults =
this.validationState.validationResults[category.name];

      // Skip categories with no issues
      if (categoryResults.issues === 0) {
        return;
      }

      htmlContent += `
        <div class="category-card">
```

```
        <div class="category-header"
onclick="toggleCategory('${category.name}')">
          <span>${this.formatCategoryName(category.name)}</span>
          <span>${categoryResults.issues - categoryResults.fixed} issues
remaining / ${categoryResults.issues} total</span>
        </div>
        <div id="${category.name}" class="category-content">
    `;

    // Add issues within category
    categoryResults.checks.forEach((check, index) => {
      if (!check.passed || check.fixed) {
        const badgeClass = check.fixed ? 'badge-fixed' :
          check.level === this.LEVELS.CRITICAL ? 'badge-critical' :
          check.level === this.LEVELS.WARNING ? 'badge-warning' :
'badge-info';
        const badgeText = check.fixed ? 'FIXED' : check.level.toUpperCase();

        htmlContent += `
          <div class="issue-item">
            <div class="issue-title">
              <span class="issue-badge ${badgeClass}">${badgeText}</span>
              ${check.name}
            </div>
            <div class="issue-message">${check.message}</div>
            ${check.details ? `
              <div>
                <a href="#"
onclick="toggleDetails('${category.name}-${index}'); return false;">Show
details</a>
                <div id="${category.name}-${index}"
class="issue-details">${check.details}</div>
              </div>
```

```
            ` : ''}
          </div>
        `;
      }
    });

    htmlContent += `
        </div>
      </div>
    `;
  });

  // Add close button and scripts
  htmlContent += `
      <button class="btn" onclick="google.script.host.close()">Close</button>
      <script>
        function toggleCategory(id) {
          var content = document.getElementById(id);
          if (content.style.display === 'block') {
            content.style.display = 'none';
          } else {
            content.style.display = 'block';
          }
        }

        function toggleDetails(id) {
          var details = document.getElementById(id);
          if (details.style.display === 'block') {
            details.style.display = 'none';
          } else {
            details.style.display = 'block';
          }
        }
```

```
        // Expand first category with issues
        document.addEventListener('DOMContentLoaded', function() {
          var categories =
document.getElementsByClassName('category-content');
          if (categories.length > 0) {
            categories[0].style.display = 'block';
          }
        });
      </script>
    </body>
    </html>
  `;

  // Show the results
  const html = HtmlService.createHtmlOutput(htmlContent)
    .setWidth(700)
    .setHeight(600);

  SpreadsheetApp.getUi().showModalDialog(html, 'System Validation Results');
},
 /**
 * Shows an error that occurred during validation
 * @param {Error} error - The error that occurred
 */
showValidationError: function(error) {
  const ui = SpreadsheetApp.getUi();

  ui.alert(
    'Validation Error',
    `An error occurred during system validation:\n${error.message}\n\nPlease
try again later.`,
    ui.ButtonSet.OK
```

```
    );
  },
  /**
   * Logs validation run to the system log
   * @param {object} summary - The validation summary
   */
  logValidationRun: function(summary) {
    try {
      logSystemActivity(
        'System Validation',
        `Ran validation: ${summary.totalChecks} checks, ${summary.issuesFound}
issues found, ${summary.issuesFixed} fixed`
      );
    } catch (e) {
      console.error('Error logging validation run:', e);
    }
  },
  /**
   * Helper method to check if arrays match
   * @param {Array} arr1 - First array
   * @param {Array} arr2 - Second array
   * @return {boolean} Whether arrays match
   */
  arraysMatch: function(arr1, arr2) {
    if (!arr1 || !arr2) return false;
    if (arr1.length !== arr2.length) return false;

    for (let i = 0; i < arr1.length; i++) {
      if (arr1[i] !== arr2[i]) return false;
    }

    return true;
  },
```

```
  /**
   * Helper method to check if email is valid
   * @param {string} email - Email to check
   * @return {boolean} Whether email is valid
   */
  isValidEmail: function(email) {
    const re =
/^(([^<>()\[\]\\.,;:\s@"]+(\.[^<>()\[\]\\.,;:\s@"]+)*)|(".+"))@((\[[0-9]{1,3}\
.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/;
    return re.test(String(email).toLowerCase());
  },
  /**
   * Helper method to format category name for display
   * @param {string} category - Category key
   * @return {string} Formatted category name
   */
  formatCategoryName: function(category) {
    switch(category) {
      case this.CATEGORIES.SHEET_STRUCTURE:
        return 'Sheet Structure';
      case this.CATEGORIES.DATA_INTEGRITY:
        return 'Data Integrity';
      case this.CATEGORIES.CONFIGURATION:
        return 'System Configuration';
      case this.CATEGORIES.ACCESS_PERMISSIONS:
        return 'Access Permissions';
      case this.CATEGORIES.FORMULA_HEALTH:
        return 'Formula Health';
      default:
        return category.replace(/_/g, ' ').replace(/\b\w/g, l =>
l.toUpperCase());
    }
  },
```

```
  /**
   * Gets all required sheets from configuration
   * @return {Array} Array of required sheet configurations
   */
  getRequiredSheets: function() {
    // Convert CRISIS_SUPPORT_CONFIG.SHEETS to array of objects with headers
    const requiredSheets = [];

    // Add standard sheets with headers
    requiredSheets.push({
      name: CRISIS_SUPPORT_CONFIG.SHEETS.COUNSELOR_TRACKING,
      critical: true,
      headers: ['Date Added', 'Name', 'Email', 'Phone', 'Status', 'Team',
'Start Date', 'Goal/Focus', 'Notes']
    });

    requiredSheets.push({
      name: CRISIS_SUPPORT_CONFIG.SHEETS.CALL_METRICS,
      critical: true,
      headers: ['Date', 'Calls Offered', 'Calls Accepted', 'Talk Time (min)',
'After Call Work (min)', 'On Queue %', 'Behaviors Impacting Performance']
    });

    requiredSheets.push({
      name: CRISIS_SUPPORT_CONFIG.SHEETS.ALERTS,
      critical: true,
      headers: ['Date', 'Severity', 'Message', 'Category', 'Created By',
'Status', 'Resolution', 'Resolved By', 'Resolved Date']
    });

    requiredSheets.push({
      name: CRISIS_SUPPORT_CONFIG.SHEETS.TASKS,
      critical: true,
```

```
    headers: ['Date Created', 'Task', 'Description', 'Assignee', 'Due Date', 'Priority', 'Status', 'Completed Date', 'Completed By']
  });

  requiredSheets.push({
    name: CRISIS_SUPPORT_CONFIG.SHEETS.SCHEDULE,
    critical: false,
    headers: ['Week Start Date', 'Work Week Start Day', 'Day 1 Date', 'Day 2 Date', 'Day 3 Date', 'Day 4 Date', 'Day 5 Date', 'Day 6 Date', 'Day 7 Date']
  });

  requiredSheets.push({
    name: CRISIS_SUPPORT_CONFIG.SHEETS.ERROR_LOG,
    critical: true,
    headers: ['Timestamp', 'Function', 'Error Type', 'Error Message', 'Stack Trace']
  });

  requiredSheets.push({
    name: CRISIS_SUPPORT_CONFIG.SHEETS.SYSTEM_LOG,
    critical: true,
    headers: ['Timestamp', 'Action', 'Details', 'User']
  });

  // Add additional sheets
  requiredSheets.push({
    name: 'Coaching Notes',
    critical: false,
    headers: ['Date', 'Counselor', 'Coach', 'Type', 'Notes', 'Follow-up Date', 'Status']
  });

  requiredSheets.push({
```

```
    name: 'One-on-One Notes',
    critical: false,
    headers: ['ID', 'Counselor Name', 'Meeting Date', 'Next Check-In Date',
'Feeling Overall', 'Work-Life Balance', 'Workload Support', 'Accomplishments',
'Feedback Reflection', 'Recent Challenges', 'Blockers', 'Growth Areas',
'Support Needed', 'Created By', 'Created Date', 'Last Modified', 'Last
Modified By']
  });

  requiredSheets.push({
    name: 'Team Lead Shifts',
    critical: false,
    headers: ['Date', 'Shift Type', 'Start Time', 'End Time', 'Goals', 'Quick
Notes', 'Created By', 'Timestamp']
  });

  requiredSheets.push({
    name: 'Time Logs',
    critical: false,
    headers: ['Date', 'Primary Shift Time', 'Admin Time', 'Meeting Time',
'1:1 Time', 'Other Time', 'Total Time', 'Activity Notes', 'Tasks
Accomplished', 'Summary Notes', 'User', 'Timestamp']
  });

  requiredSheets.push({
    name: 'Activity Breakdown',
    critical: false,
    headers: ['Date', 'User', 'Activity Type', 'Start Time', 'End Time',
'Duration (seconds)', 'Duration (formatted)', 'Notes']
  });

  requiredSheets.push({
    name: 'Time Summary',
```

```
    critical: false,
    headers: ['Week', 'User', 'Primary Shift Hours', 'Admin Hours', 'Meeting Hours', '1:1 Hours', 'Other Hours', 'Total Hours', 'Tasks Count', 'Last Updated']
  });

  return requiredSheets;
 }
};

/**
* Function to run the validation from the UI
*/
function validateSystemStructure() {
 return ValidationSystem.runValidation(true, false);
}

/**
* Function to run the validation with automatic fixes
*/
function validateAndFixSystemStructure() {
 return ValidationSystem.runValidation(true, true);
}

/**
* Function to get validation status for dashboard
*/
function getValidationStatus() {
 // Run a silent validation without fixing
 const result = ValidationSystem.runValidation(false, false);
  return {
   lastRun: result.timestamp,
   passingRate: result.passingRate,
```

```
    criticalIssues: result.criticalIssues,
    warningIssues: result.warningIssues,
    issuesFound: result.issuesFound,
    success: result.success
  };
}

/**
* Function to update the progress for the validation dialog
*/
function listenForProgressUpdates() {
  return ValidationSystem.listenForProgressUpdates();
}

/**
* Create validation menu items
*/
function addValidationMenuItems(menu) {
  return menu
    .addSeparator()
    .addSubMenu(SpreadsheetApp.getUi().createMenu('🔍 System Validation')
      .addItem('Validate System Structure', 'validateSystemStructure')
      .addItem('Validate and Auto-Fix Issues',
'validateAndFixSystemStructure'));
}
<!DOCTYPE html>
<html>
<head>
  <base target="_top">
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
 <link
href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500;700&fam
ily=Poppins:wght@500;600;700&display=swap" rel="stylesheet">
 <style>
   /* Variables for 988 Lifeline LGBTQIA+ Services color scheme */
   :root {
     /* Primary Colors */
     --trevor-orange: #FF786E;
     --deep-blue: #001A4E;
     --purple: #9A3499;
     --teal: #137F6A;

     /* Secondary Colors */
     --light-blue: #4F52DE;
     --soft-yellow: #FFAD8D;
     --lavender: #B3AE4A;
     --light-purple: #F54AC;

     /* Tertiary Colors */
     --soft-green: #BAE2CE;
     --pale-yellow: #FFF2DF;
     --light-pink: #FBCBBE;
     --soft-lavender: #D1CFCC;

     /* Gradients */
     --primary-gradient: linear-gradient(135deg, var(--trevor-orange),
var(--light-pink));
     --calm-gradient: linear-gradient(135deg, var(--light-blue),
var(--soft-green));
     --support-gradient: linear-gradient(45deg, var(--purple),
var(--light-purple));
     --background-gradient: linear-gradient(180deg, var(--pale-yellow),
#FFFFFF);
```

```
  /* Spacing */
  --spacing-xs: 4px;
  --spacing-sm: 8px;
  --spacing-md: 16px;
  --spacing-lg: 24px;
  --spacing-xl: 32px;

  /* Borders */
  --border-radius-sm: 8px;
  --border-radius-md: 12px;
  --border-radius-lg: 16px;
}

body {
  font-family: 'Roboto', sans-serif;
  margin: 0;
  padding: 0;
  background-color: var(--pale-yellow);
  color: var(--deep-blue);
}

.container {
  max-width: 800px;
  margin: 0 auto;
  padding: var(--spacing-md);
}

.header {
  margin-bottom: var(--spacing-lg);
  text-align: center;
}
```

```css
h1, h2, h3, h4 {
  font-family: 'Poppins', sans-serif;
  color: var(--deep-blue);
}

.card {
  background: white;
  border-radius: var(--border-radius-md);
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.05);
  padding: var(--spacing-lg);
  margin-bottom: var(--spacing-lg);
  position: relative;
}

.card::before {
  content: '';
  position: absolute;
  left: 0;
  top: 0;
  height: 100%;
  width: 3px;
  background: var(--trevor-orange);
}

.filter-bar {
  display: flex;
  justify-content: space-between;
  align-items: center;
  margin-bottom: var(--spacing-md);
  flex-wrap: wrap;
  gap: var(--spacing-sm);
}
```

```css
.filter-item {
  flex: 1;
  min-width: 150px;
}

.filter-label {
  font-size: 14px;
  margin-bottom: var(--spacing-xs);
  display: block;
  font-weight: 500;
}

.filter-select {
  width: 100%;
  padding: var(--spacing-xs) var(--spacing-sm);
  border: 1px solid #ddd;
  border-radius: var(--border-radius-sm);
  font-size: 14px;
}

.alert-item {
  background: white;
  border-radius: var(--border-radius-sm);
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.05);
  margin-bottom: var(--spacing-md);
  overflow: hidden;
  border-left: 4px solid var(--trevor-orange);
}

.alert-header {
  display: flex;
  justify-content: space-between;
  padding: var(--spacing-sm) var(--spacing-md);
```

```
  align-items: center;
  background-color: #f9f9f9;
  cursor: pointer;
}

.alert-title {
  font-weight: 500;
  flex: 1;
  display: flex;
  align-items: center;
  gap: var(--spacing-sm);
}

.alert-priority {
  display: inline-block;
  padding: 2px 8px;
  border-radius: 12px;
  font-size: 12px;
  margin-right: var(--spacing-sm);
}

.priority-high {
  background-color: var(--light-pink);
}

.priority-medium {
  background-color: var(--soft-yellow);
}

.priority-low {
  background-color: var(--soft-green);
}
```

```css
.alert-meta {
  font-size: 14px;
  color: #666;
}

.alert-content {
  padding: var(--spacing-md);
  display: none;
}

.alert-details {
  margin-bottom: var(--spacing-md);
}

.alert-info {
  display: grid;
  grid-template-columns: 1fr 1fr;
  gap: var(--spacing-sm) var(--spacing-md);
  margin-top: var(--spacing-md);
  font-size: 14px;
}

.alert-info-item {
  display: flex;
}

.alert-info-label {
  font-weight: 500;
  margin-right: var(--spacing-xs);
  min-width: 120px;
}

.alert-actions {
```

```
  display: flex;
  justify-content: flex-end;
  gap: var(--spacing-sm);
  padding-top: var(--spacing-md);
  border-top: 1px solid #f0f0f0;
}

.btn {
  border: none;
  padding: var(--spacing-sm) var(--spacing-md);
  border-radius: var(--border-radius-sm);
  font-weight: 500;
  cursor: pointer;
  transition: all 0.2s ease;
  font-size: 14px;
}

.btn-primary {
  background: var(--primary-gradient);
  color: white;
}

.btn-primary:hover {
  box-shadow: 0 4px 8px rgba(255, 120, 110, 0.3);
  transform: translateY(-2px);
}

.btn-secondary {
  background: white;
  color: var(--deep-blue);
  border: 1px solid #ddd;
}
```

```css
.btn-secondary:hover {
  background: #f5f5f5;
}

.empty-state {
  text-align: center;
  padding: var(--spacing-lg) var(--spacing-xl);
  color: #666;
  font-style: italic;
}

.loading {
  text-align: center;
  padding: var(--spacing-lg);
  color: #666;
}

.success-message {
  background-color: var(--soft-green);
  color: var(--deep-blue);
  padding: var(--spacing-md);
  border-radius: var(--border-radius-sm);
  margin-bottom: var(--spacing-lg);
  text-align: center;
  display: none;
}

.error-message {
  background-color: var(--light-pink);
  color: var(--deep-blue);
  padding: var(--spacing-md);
  border-radius: var(--border-radius-sm);
  margin-bottom: var(--spacing-lg);
```

```
  text-align: center;
  display: none;
}

.resolution-dialog {
  display: none;
  position: fixed;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  background-color: rgba(0, 0, 0, 0.5);
  z-index: 1000;
  justify-content: center;
  align-items: center;
}

.dialog-content {
  background: white;
  border-radius: var(--border-radius-md);
  box-shadow: 0 4px 20px rgba(0, 0, 0, 0.2);
  padding: var(--spacing-lg);
  max-width: 500px;
  width: 100%;
}

.dialog-header {
  margin-bottom: var(--spacing-md);
}

.dialog-form {
  margin-bottom: var(--spacing-md);
}
```

```
  .dialog-actions {
    display: flex;
    justify-content: flex-end;
    gap: var(--spacing-sm);
  }

  .close-button {
    position: absolute;
    top: var(--spacing-sm);
    right: var(--spacing-sm);
    background: none;
    border: none;
    font-size: 24px;
    cursor: pointer;
    color: #666;
  }

  @media (max-width: 600px) {
    .filter-bar {
      flex-direction: column;
    }

    .filter-item {
      width: 100%;
    }

    .alert-info {
      grid-template-columns: 1fr;
    }
  }
 </style>
</head>
```

```
<body>
 <div class="container">
   <div class="header">
     <h1>Active Alerts</h1>
     <p>View and manage active alerts for the 988 Lifeline LGBTQIA+ Crisis
Support System</p>
   </div>

   <div id="successMessage" class="success-message">
     Alert updated successfully!
   </div>

   <div id="errorMessage" class="error-message">
     An error occurred. Please try again.
   </div>

   <div class="card">
     <div class="filter-bar">
       <div class="filter-item">
         <label class="filter-label">Filter by Type</label>
         <select id="typeFilter" class="filter-select"
onchange="filterAlerts()">
           <option value="">All Types</option>
           <option value="System Issue">System Issue</option>
           <option value="Training Requirement">Training Requirement</option>
           <option value="Service Disruption">Service Disruption</option>
           <option value="Policy Update">Policy Update</option>
           <option value="LGBTQIA+ Resource Update">LGBTQIA+ Resource
Update</option>
           <option value="Other">Other</option>
         </select>
       </div>
```

```
      <div class="filter-item">
        <label class="filter-label">Filter by Priority</label>
        <select id="priorityFilter" class="filter-select"
onchange="filterAlerts()">
          <option value="">All Priorities</option>
          <option value="High">High</option>
          <option value="Medium">Medium</option>
          <option value="Low">Low</option>
        </select>
      </div>

      <div class="filter-item">
        <label class="filter-label">Sort By</label>
        <select id="sortOrder" class="filter-select"
onchange="filterAlerts()">
          <option value="priority">Priority (High to Low)</option>
          <option value="newest">Date (Newest First)</option>
          <option value="oldest">Date (Oldest First)</option>
        </select>
      </div>
    </div>

    <div id="alertsLoading" class="loading">
      Loading alerts...
    </div>

    <div id="alertsList">
      <!-- Alerts will be populated here -->
    </div>
  </div>
</div>
 <!-- Resolution Dialog -->
<div id="resolutionDialog" class="resolution-dialog">
```

```
    <div class="dialog-content">
      <button class="close-button"
onclick="closeResolutionDialog()">&times;</button>
      <div class="dialog-header">
        <h3>Resolve Alert</h3>
      </div>
      <div class="dialog-form">
        <div class="form-group">
          <label for="resolutionNotes">Resolution Notes</label>
          <textarea id="resolutionNotes" placeholder="Describe how this alert
was resolved..."></textarea>
        </div>
      </div>
      <div class="dialog-actions">
        <button class="btn btn-secondary"
onclick="closeResolutionDialog()">Cancel</button>
        <button class="btn btn-primary" onclick="submitResolution()">Resolve
Alert</button>
      </div>
    </div>
  </div>
   <script>
    // Global variables
    let allAlerts = [];
    let currentAlertId = null;

    // Initialize
    function initialize() {
      loadAlerts();
    }

    // Load alerts
    function loadAlerts() {
```

```
    document.getElementById('alertsLoading').style.display = 'block';
    document.getElementById('alertsList').innerHTML = '';

    google.script.run
      .withSuccessHandler(function(alerts) {
        allAlerts = alerts;
        document.getElementById('alertsLoading').style.display = 'none';

        if (alerts && alerts.length > 0) {
          filterAlerts(); // This will render the alerts with current filters
        } else {
          document.getElementById('alertsList').innerHTML =
            '<div class="empty-state">No active alerts found.</div>';
        }
      })
      .withFailureHandler(function(error) {
        document.getElementById('alertsLoading').style.display = 'none';
        document.getElementById('alertsList').innerHTML =
          '<div class="empty-state">Error loading alerts: ' + error +
'</div>';
      })
      .getActiveAlerts();
  }

  // Filter and sort alerts
  function filterAlerts() {
    const typeFilter = document.getElementById('typeFilter').value;
    const priorityFilter = document.getElementById('priorityFilter').value;
    const sortOrder = document.getElementById('sortOrder').value;

    // Apply filters
    let filteredAlerts = [...allAlerts];
```

```
    if (typeFilter) {
      filteredAlerts = filteredAlerts.filter(alert => alert.type === 
typeFilter);
    }

    if (priorityFilter) {
      filteredAlerts = filteredAlerts.filter(alert => alert.priority === 
priorityFilter);
    }

    // Apply sorting
    if (sortOrder === 'priority') {
      const priorityOrder = { 'High': 0, 'Medium': 1, 'Low': 2 };
      filteredAlerts.sort((a, b) => {
        return priorityOrder[a.priority] - priorityOrder[b.priority];
      });
    } else if (sortOrder === 'newest') {
      filteredAlerts.sort((a, b) => new Date(b.date) - new Date(a.date));
    } else if (sortOrder === 'oldest') {
      filteredAlerts.sort((a, b) => new Date(a.date) - new Date(b.date));
    }

    // Render alerts
    renderAlerts(filteredAlerts);
  }

  // Render the alerts
  function renderAlerts(alerts) {
    const listElement = document.getElementById('alertsList');
    listElement.innerHTML = '';

    if (alerts.length === 0) {
```

```
      listElement.innerHTML = '<div class="empty-state">No alerts match the
selected filters.</div>';
      return;
    }

    alerts.forEach(function(alert) {
      const alertItem = document.createElement('div');
      alertItem.className = 'alert-item';
      alertItem.setAttribute('data-alert-id', alert.id);

      // Determine priority class
      let priorityClass = 'priority-medium';
      if (alert.priority === 'High') {
        priorityClass = 'priority-high';
      } else if (alert.priority === 'Low') {
        priorityClass = 'priority-low';
      }

      // Format date
      const dateStr = new Date(alert.date).toLocaleDateString();

      // Create the alert header
      const alertHeader = document.createElement('div');
      alertHeader.className = 'alert-header';
      alertHeader.innerHTML = `
        <div class="alert-title">
          <span class="alert-priority
${priorityClass}">${alert.priority}</span>
          ${alert.type}
        </div>
        <div class="alert-meta">${dateStr}</div>
      `;
      alertHeader.addEventListener('click', function() {
```

```
      toggleAlertContent(alert.id);
    });

    // Create the alert content
    const alertContent = document.createElement('div');
    alertContent.className = 'alert-content';
    alertContent.id = `alert-content-${alert.id}`;

    // Format expiration date if exists
    let expirationStr = 'No expiration';
    if (alert.expirationDate) {
      expirationStr = new Date(alert.expirationDate).toLocaleDateString();
    }

    // Create the content HTML
    alertContent.innerHTML = `
      <div class="alert-details">
        ${alert.message}
      </div>
      <div class="alert-info">
        <div class="alert-info-item">
          <div class="alert-info-label">Created By:</div>
          <div>${alert.createdBy || 'Unknown'}</div>
        </div>
        <div class="alert-info-item">
          <div class="alert-info-label">Assigned To:</div>
          <div>${alert.assignedTo || 'All Team Members'}</div>
        </div>
        <div class="alert-info-item">
          <div class="alert-info-label">Expires:</div>
          <div>${expirationStr}</div>
        </div>
        <div class="alert-info-item">
```

```
            <div class="alert-info-label">Status:</div>
            <div>${alert.status || 'Active'}</div>
          </div>
        </div>
        <div class="alert-actions">
          <button class="btn btn-secondary"
onclick="snoozeAlert('${alert.id}')">Snooze (24h)</button>
          <button class="btn btn-primary"
onclick="showResolutionDialog('${alert.id}')">Resolve</button>
        </div>
      `;

      // Add header and content to the alert item
      alertItem.appendChild(alertHeader);
      alertItem.appendChild(alertContent);

      // Add to the list
      listElement.appendChild(alertItem);
    });
  }

  // Toggle alert content visibility
  function toggleAlertContent(alertId) {
    const contentElement =
document.getElementById(`alert-content-${alertId}`);
    if (contentElement.style.display === 'block') {
      contentElement.style.display = 'none';
    } else {
      // Hide all other expanded alerts
      document.querySelectorAll('.alert-content').forEach(function(content) {
        content.style.display = 'none';
      });
```

```
        // Show this alert
        contentElement.style.display = 'block';
      }
    }

    // Snooze an alert for 24 hours
    function snoozeAlert(alertId) {
      // Hide messages
      document.getElementById('successMessage').style.display = 'none';
      document.getElementById('errorMessage').style.display = 'none';

      google.script.run
        .withSuccessHandler(function(result) {
          if (result.success) {
            // Show success message
            document.getElementById('successMessage').textContent = 'Alert
snoozed for 24 hours.';
            document.getElementById('successMessage').style.display = 'block';

            // Reload alerts
            loadAlerts();
          } else {
            // Show error message
            document.getElementById('errorMessage').textContent =
result.message || 'Failed to snooze alert.';
            document.getElementById('errorMessage').style.display = 'block';
          }
        })
        .withFailureHandler(function(error) {
          document.getElementById('errorMessage').textContent = 'Error: ' +
error;
          document.getElementById('errorMessage').style.display = 'block';
        })
```

```
    .snoozeAlert(alertId);
}

// Show resolution dialog
function showResolutionDialog(alertId) {
  currentAlertId = alertId;
  document.getElementById('resolutionNotes').value = '';
  document.getElementById('resolutionDialog').style.display = 'flex';
}

// Close resolution dialog
function closeResolutionDialog() {
  document.getElementById('resolutionDialog').style.display = 'none';
  currentAlertId = null;
}

// Submit resolution
function submitResolution() {
  // Validate
  const notes = document.getElementById('resolutionNotes').value;
  if (!notes) {
    alert('Please enter resolution notes');
    return;
  }

  // Hide messages
  document.getElementById('successMessage').style.display = 'none';
  document.getElementById('errorMessage').style.display = 'none';

  google.script.run
    .withSuccessHandler(function(result) {
      if (result.success) {
        // Close dialog
```

```
            closeResolutionDialog();

            // Show success message
            document.getElementById('successMessage').textContent = 'Alert
resolved successfully.';
            document.getElementById('successMessage').style.display = 'block';

            // Reload alerts
            loadAlerts();
          } else {
            // Show error message
            document.getElementById('errorMessage').textContent =
result.message || 'Failed to resolve alert.';
            document.getElementById('errorMessage').style.display = 'block';
            closeResolutionDialog();
          }
        })
        .withFailureHandler(function(error) {
          document.getElementById('errorMessage').textContent = 'Error: ' +
error;
          document.getElementById('errorMessage').style.display = 'block';
          closeResolutionDialog();
        })
        .resolveAlert(currentAlertId, notes);
    }

    // Initialize when the page loads
    google.script.onload = function() {
      initialize();
    };
  </script>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<head>
 <base target="_top">
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <link
href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500;700&fam
ily=Poppins:wght@500;600;700&display=swap" rel="stylesheet">
 <style>
   :root {
     /* Primary Colors */
     --trevor-orange: #FF786E;
     --deep-blue: #001A4E;
     --purple: #9A3499;
     --teal: #137F6A;

     /* Secondary Colors */
     --light-blue: #4F52DE;
     --soft-yellow: #FFAD8D;
     --light-purple: #F54AC;

     /* Tertiary Colors */
     --soft-green: #BAE2CE;
     --pale-yellow: #FFF2DF;
     --light-pink: #FBCBBE;

     /* Gradients */
     --primary-gradient: linear-gradient(135deg, var(--trevor-orange),
var(--light-pink));
     --background-gradient: linear-gradient(180deg, var(--pale-yellow),
#FFFFFF);
```

```
  /* Spacing */
  --spacing-xs: 4px;
  --spacing-sm: 8px;
  --spacing-md: 16px;
  --spacing-lg: 24px;
  --spacing-xl: 32px;

  /* Borders */
  --border-radius-sm: 8px;
  --border-radius-md: 12px;
  --border-radius-lg: 16px;
}

body {
  font-family: 'Roboto', sans-serif;
  background: var(--background-gradient);
  color: var(--deep-blue);
  margin: 0;
  padding: 0;
  min-height: 100vh;
}

.container {
  max-width: 800px;
  margin: 0 auto;
  padding: var(--spacing-lg);
}

h1, h2, h3 {
  font-family: 'Poppins', sans-serif;
  color: var(--deep-blue);
}
```

```css
.header {
  margin-bottom: var(--spacing-xl);
  position: relative;
  padding-bottom: var(--spacing-md);
  border-bottom: 1px solid rgba(0, 0, 0, 0.1);
}

.header h1 {
  margin-bottom: var(--spacing-xs);
}

.header p {
  color: #666;
  margin-top: 0;
}

.card {
  background: white;
  border-radius: var(--border-radius-md);
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.05);
  overflow: hidden;
  position: relative;
  margin-bottom: var(--spacing-xl);
}

.card::before {
  content: '';
  position: absolute;
  left: 0;
  top: 0;
  height: 100%;
  width: 3px;
  background: var(--trevor-orange);
```

```css
}

.card-content {
  padding: var(--spacing-lg);
}

.form-group {
  margin-bottom: var(--spacing-lg);
}

.form-row {
  display: flex;
  flex-wrap: wrap;
  margin: 0 -10px;
}

.form-col {
  flex: 1;
  padding: 0 10px;
  min-width: 200px;
}

label {
  display: block;
  margin-bottom: var(--spacing-xs);
  font-weight: 500;
}

input, select, textarea {
  width: 100%;
  padding: 10px 12px;
  border: 1px solid #ddd;
  border-radius: var(--border-radius-sm);
```

```
  font-family: 'Roboto', sans-serif;
  font-size: 14px;
  transition: all 0.2s ease;
}

input:focus, select:focus, textarea:focus {
  outline: none;
  border-color: var(--trevor-orange);
  box-shadow: 0 0 0 3px rgba(255, 120, 110, 0.2);
}

textarea {
  min-height: 120px;
  resize: vertical;
}

.required::after {
  content: '*';
  color: var(--trevor-orange);
  margin-left: 2px;
}

.help-text {
  font-size: 12px;
  color: #666;
  margin-top: var(--spacing-xs);
}

.btn {
  border: none;
  padding: var(--spacing-sm) var(--spacing-lg);
  border-radius: var(--border-radius-sm);
  font-weight: 500;
```

```
  cursor: pointer;
  transition: all 0.2s ease;
  font-family: 'Poppins', sans-serif;
}

.btn-primary {
  background: var(--primary-gradient);
  color: white;
}

.btn-primary:hover {
  box-shadow: 0 4px 8px rgba(255, 120, 110, 0.3);
  transform: translateY(-2px);
}

.btn-secondary {
  background: white;
  color: var(--deep-blue);
  border: 1px solid #ddd;
}

.btn-secondary:hover {
  background: #f5f5f5;
}

.form-actions {
  display: flex;
  justify-content: space-between;
  margin-top: var(--spacing-xl);
}

.notification {
  padding: var(--spacing-md);
```

```
  border-radius: var(--border-radius-sm);
  margin-bottom: var(--spacing-lg);
  display: none;
}

.notification.success {
  background-color: rgba(19, 127, 106, 0.1);
  color: var(--teal);
  border-left: 3px solid var(--teal);
}

.notification.error {
  background-color: rgba(245, 74, 12, 0.1);
  color: var(--light-purple);
  border-left: 3px solid var(--light-purple);
}

.field-error {
  color: var(--light-purple);
  font-size: 12px;
  margin-top: var(--spacing-xs);
  display: none;
}

@media (max-width: 600px) {
  .form-col {
    flex: 100%;
    margin-bottom: var(--spacing-md);
  }

  .form-actions {
    flex-direction: column;
  }
```

```
    .form-actions button {
      margin-bottom: var(--spacing-sm);
    }
  }
 </style>
</head>
<body>
 <div class="container">
  <div class="header">
    <h1>Add New Counselor</h1>
    <p>Add a new counselor to your team for the 988 Lifeline LGBTQIA+
Services</p>
  </div>

  <div id="notification" class="notification"></div>

  <div class="card">
    <div class="card-content">
      <form id="counselorForm">
        <div class="form-row">
          <div class="form-col">
            <div class="form-group">
              <label for="name" class="required">Full Name</label>
              <input type="text" id="name" name="name" required>
              <div id="nameError" class="field-error">Please enter the
counselor's name</div>
            </div>
          </div>

          <div class="form-col">
            <div class="form-group">
              <label for="email" class="required">Email Address</label>
```

```
              <input type="email" id="email" name="email" required>
              <div id="emailError" class="field-error">Please enter a valid email address</div>
            </div>
          </div>

          <div class="form-row">
            <div class="form-col">
              <div class="form-group">
                <label for="phone">Phone Number</label>
                <input type="tel" id="phone" name="phone">
                <div id="phoneError" class="field-error">Please enter a valid phone number</div>
              </div>
            </div>

            <div class="form-col">
              <div class="form-group">
                <label for="status" class="required">Status</label>
                <select id="status" name="status" required>
                  <option value="">Select Status</option>
                  <option value="Active">Active</option>
                  <option value="Training">Training</option>
                  <option value="PTO">Planned Time Off</option>
                  <option value="LOA">Leave of Absence</option>
                  <option value="Inactive">Inactive</option>
                </select>
                <div id="statusError" class="field-error">Please select a status</div>
              </div>
            </div>
          </div>
```

```
        <div class="form-row">
          <div class="form-col">
            <div class="form-group">
              <label for="team" class="required">Team</label>
              <select id="team" name="team" required>
                <option value="">Select Team</option>
                <option value="Digital">Digital</option>
                <option value="Lifeline">Lifeline</option>
                <option value="Hybrid">Hybrid</option>
              </select>
              <div id="teamError" class="field-error">Please select a
team</div>
            </div>
          </div>

          <div class="form-col">
            <div class="form-group">
              <label for="startDate" class="required">Start Date</label>
              <input type="date" id="startDate" name="startDate" required>
              <div id="startDateError" class="field-error">Please select a
start date</div>
            </div>
          </div>
        </div>

        <div class="form-group">
          <label for="goalFocus">Goal/Focus</label>
          <textarea id="goalFocus" name="goalFocus" placeholder="Enter
initial goals and focus areas for this counselor..."></textarea>
        </div>

        <div class="form-group">
```

```
            <label for="notes">Notes</label>
            <textarea id="notes" name="notes" placeholder="Enter any additional
notes about this counselor..."></textarea>
          </div>

          <div class="form-actions">
            <button type="button" class="btn btn-secondary"
onclick="google.script.host.close()">Cancel</button>
            <div>
              <button type="button" class="btn btn-secondary"
onclick="clearForm()">Clear Form</button>
              <button type="submit" class="btn btn-primary">Add
Counselor</button>
            </div>
          </div>
        </form>
      </div>
    </div>
  </div>
  <script>
    // Initialize form
    document.addEventListener('DOMContentLoaded', function() {
      document.getElementById('counselorForm').addEventListener('submit',
submitForm);
    });

    // Form submission
    function submitForm(e) {
      e.preventDefault();

      // Reset all error messages
      const errorElements = document.querySelectorAll('.field-error');
      errorElements.forEach(element => {
```

```
    element.style.display = 'none';
  });

  // Validate form
  let isValid = true;

  // Required fields
  const requiredFields = ['name', 'email', 'status', 'team', 'startDate'];
  requiredFields.forEach(field => {
    const element = document.getElementById(field);
    if (!element.value.trim()) {
      document.getElementById(`${field}Error`).style.display = 'block';
      isValid = false;
    }
  });

  // Email validation
  const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
  const email = document.getElementById('email');
  if (email.value && !emailRegex.test(email.value)) {
    document.getElementById('emailError').style.display = 'block';
    isValid = false;
  }

  // Phone validation (if provided)
  const phone = document.getElementById('phone');
  if (phone.value && !/^[\d\s\(\)\-\+]+$/.test(phone.value)) {
    document.getElementById('phoneError').style.display = 'block';
    isValid = false;
  }

  if (!isValid) {
    return;
```

```
  }

  // Collect form data
  const formData = {
    name: document.getElementById('name').value,
    email: document.getElementById('email').value,
    phone: document.getElementById('phone').value,
    status: document.getElementById('status').value,
    team: document.getElementById('team').value,
    startDate: document.getElementById('startDate').value,
    goalFocus: document.getElementById('goalFocus').value,
    notes: document.getElementById('notes').value
  };

  // Submit to Google Apps Script
  google.script.run
    .withSuccessHandler(onSuccess)
    .withFailureHandler(onFailure)
    .addCounselor(formData);
}

// Success handler
function onSuccess(response) {
  const notification = document.getElementById('notification');
  notification.textContent = 'Counselor added successfully!';
  notification.className = 'notification success';
  notification.style.display = 'block';

  // Clear form after success
  clearForm();

  // Scroll to notification
  notification.scrollIntoView({ behavior: 'smooth' });
```

```
    // Hide notification after 5 seconds
    setTimeout(function() {
      notification.style.display = 'none';
    }, 5000);
  }

  // Failure handler
  function onFailure(error) {
    const notification = document.getElementById('notification');
    notification.textContent = 'Error: ' + error.message;
    notification.className = 'notification error';
    notification.style.display = 'block';

    // Scroll to notification
    notification.scrollIntoView({ behavior: 'smooth' });
  }

  // Clear form fields
  function clearForm() {
    document.getElementById('counselorForm').reset();

    // Reset all error messages
    const errorElements = document.querySelectorAll('.field-error');
    errorElements.forEach(element => {
      element.style.display = 'none';
    });
  }
 </script>
</body>
</html>
<!DOCTYPE html>
<html>
```

```
<head>
 <base target="_top">
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <link
href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500;700&fam
ily=Poppins:wght@500;600;700&display=swap" rel="stylesheet">
 <style>
   /* Variables for 988 Lifeline LGBTQIA+ Services color scheme */
   :root {
     /* Primary Colors */
     --trevor-orange: #FF786E;
     --deep-blue: #001A4E;
     --purple: #9A3499;
     --teal: #137F6A;

     /* Secondary Colors */
     --light-blue: #4F52DE;
     --soft-yellow: #FFAD8D;
     --lavender: #B3AE4A;
     --light-purple: #F54AC;

     /* Tertiary Colors */
     --soft-green: #BAE2CE;
     --pale-yellow: #FFF2DF;
     --light-pink: #FBCBBE;
     --soft-lavender: #D1CFCC;

     /* Gradients */
     --primary-gradient: linear-gradient(135deg, var(--trevor-orange),
var(--light-pink));
     --calm-gradient: linear-gradient(135deg, var(--light-blue),
var(--soft-green));
```

```
    --support-gradient: linear-gradient(45deg, var(--purple),
var(--light-purple));
    --background-gradient: linear-gradient(180deg, var(--pale-yellow),
#FFFFFF);

    /* Spacing */
    --spacing-xs: 4px;
    --spacing-sm: 8px;
    --spacing-md: 16px;
    --spacing-lg: 24px;
    --spacing-xl: 32px;

    /* Borders */
    --border-radius-sm: 8px;
    --border-radius-md: 12px;
    --border-radius-lg: 16px;
  }

  body {
    font-family: 'Roboto', sans-serif;
    margin: 0;
    padding: 0;
    background-color: var(--pale-yellow);
    color: var(--deep-blue);
  }

  .container {
    max-width: 600px;
    margin: 0 auto;
    padding: var(--spacing-md);
  }

  .header {
```

```
  margin-bottom: var(--spacing-lg);
  text-align: center;
}

h1, h2, h3, h4 {
  font-family: 'Poppins', sans-serif;
  color: var(--deep-blue);
}

.form-card {
  background: white;
  border-radius: var(--border-radius-md);
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.05);
  padding: var(--spacing-lg);
  position: relative;
}

.form-card::before {
  content: '';
  position: absolute;
  left: 0;
  top: 0;
  height: 100%;
  width: 3px;
  background: var(--trevor-orange);
}

.form-group {
  margin-bottom: var(--spacing-md);
}

label {
  display: block;
```

```
  font-weight: 500;
  margin-bottom: var(--spacing-xs);
}

input, select, textarea {
  width: 100%;
  padding: var(--spacing-sm);
  border: 1px solid #ddd;
  border-radius: var(--border-radius-sm);
  font-family: 'Roboto', sans-serif;
  font-size: 16px;
}

textarea {
  resize: vertical;
  min-height: 100px;
}

.required-field::after {
  content: " *";
  color: var(--trevor-orange);
}

.btn {
  border: none;
  padding: var(--spacing-md) var(--spacing-lg);
  border-radius: var(--border-radius-sm);
  font-weight: 500;
  cursor: pointer;
  transition: all 0.2s ease;
}

.btn-primary {
```

```css
  background: var(--primary-gradient);
  color: white;
}

.btn-primary:hover {
  box-shadow: 0 4px 8px rgba(255, 120, 110, 0.3);
  transform: translateY(-2px);
}

.btn-secondary {
  background: white;
  color: var(--deep-blue);
  border: 1px solid #ddd;
}

.btn-secondary:hover {
  background: #f5f5f5;
}

.form-actions {
  display: flex;
  justify-content: space-between;
  margin-top: var(--spacing-lg);
}

.success-message {
  background-color: var(--soft-green);
  color: var(--deep-blue);
  padding: var(--spacing-md);
  border-radius: var(--border-radius-sm);
  margin-bottom: var(--spacing-lg);
  text-align: center;
  display: none;
```

```css
}

.error-message {
  background-color: var(--light-pink);
  color: var(--deep-blue);
  padding: var(--spacing-md);
  border-radius: var(--border-radius-sm);
  margin-bottom: var(--spacing-lg);
  text-align: center;
  display: none;
}

.priority-options {
  display: flex;
  gap: var(--spacing-sm);
  margin-top: var(--spacing-xs);
}

.priority-option {
  flex: 1;
  padding: var(--spacing-sm);
  border-radius: var(--border-radius-sm);
  text-align: center;
  cursor: pointer;
  transition: all 0.2s;
  border: 2px solid transparent;
}

.priority-option:hover {
  transform: translateY(-2px);
}

.priority-option.selected {
```

```
    border-color: var(--deep-blue);
  }

  .option-high {
    background-color: var(--light-pink);
  }

  .option-medium {
    background-color: var(--soft-yellow);
  }

  .option-low {
    background-color: var(--soft-green);
  }

  .help-text {
    font-size: 14px;
    color: #666;
    margin-top: var(--spacing-xs);
  }
 </style>
</head>
<body>
 <div class="container">
  <div class="header">
    <h1>Create Alert</h1>
    <p>Add an alert to the 988 Lifeline LGBTQIA+ Crisis Support System</p>
  </div>

  <div id="successMessage" class="success-message">
    Alert created successfully!
  </div>
```

```
  <div id="errorMessage" class="error-message">
    An error occurred. Please try again.
  </div>

  <div class="form-card">
    <form id="alertForm">
      <div class="form-group">
        <label for="alertType" class="required-field">Alert Type</label>
        <select id="alertType" name="alertType" required>
          <option value="">-- Select Alert Type --</option>
          <option value="System Issue">System Issue</option>
          <option value="Training Requirement">Training Requirement</option>
          <option value="Service Disruption">Service Disruption</option>
          <option value="Policy Update">Policy Update</option>
          <option value="LGBTQIA+ Resource Update">LGBTQIA+ Resource
Update</option>
          <option value="Other">Other</option>
        </select>
      </div>

      <div class="form-group">
        <label for="alertMessage" class="required-field">Alert
Message</label>
        <textarea id="alertMessage" name="alertMessage" placeholder="Describe
the alert in detail..." required></textarea>
      </div>

      <div class="form-group">
        <label class="required-field">Priority</label>
        <div class="priority-options">
          <div class="priority-option option-high" data-priority="High"
onclick="selectPriority('High')">
            High
```

```
            </div>
            <div class="priority-option option-medium" data-priority="Medium"
onclick="selectPriority('Medium')">
              Medium
            </div>
            <div class="priority-option option-low" data-priority="Low"
onclick="selectPriority('Low')">
              Low
            </div>
          </div>
          <input type="hidden" id="alertPriority" name="alertPriority"
value="">
        </div>

        <div class="form-group">
          <label for="assignedTo">Assigned To</label>
          <select id="assignedTo" name="assignedTo">
            <option value="">-- Unassigned --</option>
            <!-- Team members will be populated here -->
          </select>
          <div class="help-text">Leave unassigned if this alert applies to all
team members</div>
        </div>

        <div class="form-group">
          <label for="expirationDate">Expiration Date</label>
          <input type="date" id="expirationDate" name="expirationDate">
          <div class="help-text">Leave blank if the alert should not expire
automatically</div>
        </div>

        <div class="form-actions">
```

```
        <button type="button" class="btn btn-secondary"
onclick="cancelForm()">Cancel</button>
        <button type="submit" class="btn btn-primary">Create Alert</button>
      </div>
    </form>
  </div>
 </div>
  <script>
   // Global variables
   let selectedPriority = null;

   // Initialize the form
   function initialize() {
     // Load team members for assignment dropdown
     loadTeamMembers();

     // Set up form submission
     document.getElementById('alertForm').addEventListener('submit',
submitForm);

     // Set default expiration date to 7 days from now
     const expirationDate = new Date();
     expirationDate.setDate(expirationDate.getDate() + 7);
     document.getElementById('expirationDate').value =
expirationDate.toISOString().split('T')[0];
   }

   // Load team members for assignment dropdown
   function loadTeamMembers() {
     google.script.run
       .withSuccessHandler(function(members) {
         const select = document.getElementById('assignedTo');
```

```
        members.forEach(function(member) {
          const option = document.createElement('option');
          option.value = member.email;
          option.textContent = member.name;
          select.appendChild(option);
        });
      })
      .withFailureHandler(function(error) {
        console.error('Failed to load team members:', error);
      })
      .getTeamMembers();
}

// Select priority level
function selectPriority(priority) {
  selectedPriority = priority;

  // Update hidden input
  document.getElementById('alertPriority').value = priority;

  // Update UI
  document.querySelectorAll('.priority-option').forEach(function(option) {
    option.classList.remove('selected');
    if (option.getAttribute('data-priority') === priority) {
      option.classList.add('selected');
    }
  });
}

// Submit form
function submitForm(event) {
  event.preventDefault();
```

```
    // Hide messages
    document.getElementById('successMessage').style.display = 'none';
    document.getElementById('errorMessage').style.display = 'none';

    // Validate required fields
    const alertType = document.getElementById('alertType').value;
    const alertMessage = document.getElementById('alertMessage').value;

    if (!alertType) {
      document.getElementById('errorMessage').textContent = 'Please select an
alert type.';
      document.getElementById('errorMessage').style.display = 'block';
      return;
    }

    if (!alertMessage) {
      document.getElementById('errorMessage').textContent = 'Please enter an
alert message.';
      document.getElementById('errorMessage').style.display = 'block';
      return;
    }

    if (!selectedPriority) {
      document.getElementById('errorMessage').textContent = 'Please select a
priority level.';
      document.getElementById('errorMessage').style.display = 'block';
      return;
    }

    // Get form data
    const alertData = {
      type: alertType,
      message: alertMessage,
```

```
      priority: selectedPriority,
      assignedTo: document.getElementById('assignedTo').value,
      expirationDate: document.getElementById('expirationDate').value
    };

    // Submit to server
    google.script.run
      .withSuccessHandler(function(result) {
        if (result.success) {
          // Show success message
          document.getElementById('successMessage').textContent =
result.message || 'Alert created successfully!';
          document.getElementById('successMessage').style.display = 'block';

          // Reset form
          document.getElementById('alertForm').reset();

          // Reset priority selection

document.querySelectorAll('.priority-option').forEach(function(option) {
            option.classList.remove('selected');
          });
          selectedPriority = null;

          // Set default expiration date again
          const expirationDate = new Date();
          expirationDate.setDate(expirationDate.getDate() + 7);
          document.getElementById('expirationDate').value =
expirationDate.toISOString().split('T')[0];
        } else {
          // Show error message
          document.getElementById('errorMessage').textContent =
result.message || 'An error occurred while creating the alert.';
```

```
            document.getElementById('errorMessage').style.display = 'block';
          }
        })
        .withFailureHandler(function(error) {
          // Show error message
          document.getElementById('errorMessage').textContent = 'Error creating
alert: ' + error;
          document.getElementById('errorMessage').style.display = 'block';
        })
        .createAlert(alertData);
    }

    // Cancel form
    function cancelForm() {
      google.script.host.close();
    }

    // Initialize when the page loads
    google.script.onload = function() {
      initialize();
    };
  </script>
</body>
</html>
<!DOCTYPE html>
<html>
<head>
  <base target="_top">
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link
href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500;700&fam
ily=Poppins:wght@500;600;700&display=swap" rel="stylesheet">
```

```
<style>
  /* Variables for 988 Lifeline LGBTQIA+ Services color scheme */
  :root {
    /* Primary Colors */
    --trevor-orange: #FF786E;
    --deep-blue: #001A4E;
    --purple: #9A3499;
    --teal: #137F6A;

    /* Secondary Colors */
    --light-blue: #4F52DE;
    --soft-yellow: #FFAD8D;
    --lavender: #B3AE4A;
    --light-purple: #F54AC;

    /* Tertiary Colors */
    --soft-green: #BAE2CE;
    --pale-yellow: #FFF2DF;
    --light-pink: #FBCBBE;
    --soft-lavender: #D1CFCC;

    /* Gradients */
    --primary-gradient: linear-gradient(135deg, var(--trevor-orange),
var(--light-pink));
    --calm-gradient: linear-gradient(135deg, var(--light-blue),
var(--soft-green));
    --support-gradient: linear-gradient(45deg, var(--purple),
var(--light-purple));
    --background-gradient: linear-gradient(180deg, var(--pale-yellow),
#FFFFFF);

    /* Spacing */
    --spacing-xs: 4px;
```

```
  --spacing-sm: 8px;
  --spacing-md: 16px;
  --spacing-lg: 24px;
  --spacing-xl: 32px;

  /* Borders */
  --border-radius-sm: 8px;
  --border-radius-md: 12px;
  --border-radius-lg: 16px;
}

body {
  font-family: 'Roboto', sans-serif;
  margin: 0;
  padding: 0;
  background-color: var(--pale-yellow);
  color: var(--deep-blue);
}

.container {
  max-width: 600px;
  margin: 0 auto;
  padding: var(--spacing-md);
}

.header {
  margin-bottom: var(--spacing-lg);
  text-align: center;
}

h1, h2, h3, h4 {
  font-family: 'Poppins', sans-serif;
  color: var(--deep-blue);
```

```
}

.form-card {
  background: white;
  border-radius: var(--border-radius-md);
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.05);
  padding: var(--spacing-lg);
  position: relative;
}

.form-card::before {
  content: '';
  position: absolute;
  left: 0;
  top: 0;
  height: 100%;
  width: 3px;
  background: var(--teal);
}

.form-group {
  margin-bottom: var(--spacing-md);
}

label {
  display: block;
  font-weight: 500;
  margin-bottom: var(--spacing-xs);
}

input, select, textarea {
  width: 100%;
  padding: var(--spacing-sm);
```

```
  border: 1px solid #ddd;
  border-radius: var(--border-radius-sm);
  font-family: 'Roboto', sans-serif;
  font-size: 16px;
}

textarea {
  resize: vertical;
  min-height: 100px;
}

.required-field::after {
  content: " *";
  color: var(--trevor-orange);
}

.btn {
  border: none;
  padding: var(--spacing-md) var(--spacing-lg);
  border-radius: var(--border-radius-sm);
  font-weight: 500;
  cursor: pointer;
  transition: all 0.2s ease;
}

.btn-primary {
  background: var(--primary-gradient);
  color: white;
}

.btn-primary:hover {
  box-shadow: 0 4px 8px rgba(255, 120, 110, 0.3);
  transform: translateY(-2px);
```

```css
}

.btn-secondary {
  background: white;
  color: var(--deep-blue);
  border: 1px solid #ddd;
}

.btn-secondary:hover {
  background: #f5f5f5;
}

.form-actions {
  display: flex;
  justify-content: space-between;
  margin-top: var(--spacing-lg);
}

.success-message {
  background-color: var(--soft-green);
  color: var(--deep-blue);
  padding: var(--spacing-md);
  border-radius: var(--border-radius-sm);
  margin-bottom: var(--spacing-lg);
  text-align: center;
  display: none;
}

.error-message {
  background-color: var(--light-pink);
  color: var(--deep-blue);
  padding: var(--spacing-md);
  border-radius: var(--border-radius-sm);
```

```
  margin-bottom: var(--spacing-lg);
  text-align: center;
  display: none;
}

.help-text {
  font-size: 14px;
  color: #666;
  margin-top: var(--spacing-xs);
}

.form-row {
  display: flex;
  gap: var(--spacing-md);
}

.form-col {
  flex: 1;
}

.asana-info {
  background-color: #f8f8f8;
  padding: var(--spacing-md);
  border-radius: var(--border-radius-sm);
  margin-bottom: var(--spacing-lg);
}

.asana-logo {
  text-align: center;
  margin-bottom: var(--spacing-md);
}

.asana-logo img {
```

```
  height: 40px;
}

.tags-input {
  display: flex;
  flex-wrap: wrap;
  gap: var(--spacing-xs);
  padding: var(--spacing-xs);
  border: 1px solid #ddd;
  border-radius: var(--border-radius-sm);
  min-height: 40px;
}

.tag {
  display: inline-flex;
  align-items: center;
  background-color: var(--soft-lavender);
  padding: 2px 8px;
  border-radius: 12px;
  font-size: 14px;
}

.tag-remove {
  margin-left: 4px;
  cursor: pointer;
  font-weight: bold;
}

.tags-input input {
  flex: 1;
  min-width: 100px;
  border: none;
  padding: 4px;
```

```
    outline: none;
  }

  @media (max-width: 600px) {
    .form-row {
      flex-direction: column;
      gap: 0;
    }
  }
 </style>
</head>
<body>
 <div class="container">
  <div class="header">
    <h1>Create Asana Task</h1>
    <p>Create a task in Asana from the 988 Lifeline LGBTQIA+ Crisis Support System</p>
  </div>

  <div id="successMessage" class="success-message">
    Task created successfully in Asana!
  </div>

  <div id="errorMessage" class="error-message">
    An error occurred. Please try again.
  </div>

  <div class="asana-info">
    <div class="asana-logo">
      <img src="https://asana.com/assets/img/logo/asana-logo.svg" alt="Asana Logo">
    </div>
```

```
    <p>This form will create a task in your Asana workspace. All required Asana API credentials must be configured in Settings.</p>
  </div>

  <div class="form-card">
    <form id="asanaTaskForm">
      <div class="form-group">
        <label for="taskName" class="required-field">Task Name</label>
        <input type="text" id="taskName" name="taskName" placeholder="Enter a clear task name..." required>
      </div>

      <div class="form-group">
        <label for="taskNotes">Description</label>
        <textarea id="taskNotes" name="taskNotes" placeholder="Describe the task in detail..."></textarea>
      </div>

      <div class="form-row">
        <div class="form-col">
          <div class="form-group">
            <label for="projectSelect" class="required-field">Asana Project</label>
            <select id="projectSelect" name="projectSelect" required>
              <option value="">-- Loading Projects --</option>
            </select>
            <div class="help-text">Select the Asana project to add this task to</div>
          </div>
        </div>

        <div class="form-col">
          <div class="form-group">
```

```
            <label for="assigneeSelect">Assignee</label>
            <select id="assigneeSelect" name="assigneeSelect">
              <option value="">-- Loading Users --</option>
            </select>
            <div class="help-text">Leave blank for unassigned</div>
          </div>
        </div>
      </div>

      <div class="form-row">
        <div class="form-col">
          <div class="form-group">
            <label for="dueDate">Due Date</label>
            <input type="date" id="dueDate" name="dueDate">
          </div>
        </div>

        <div class="form-col">
          <div class="form-group">
            <label for="sectionSelect">Section</label>
            <select id="sectionSelect" name="sectionSelect">
              <option value="">-- Select Project First --</option>
            </select>
          </div>
        </div>
      </div>

      <div class="form-group">
        <label for="tagsInput">Tags</label>
        <div id="tagsContainer" class="tags-input">
          <input type="text" id="tagsInput" placeholder="Type tag and press
Enter" onkeydown="handleTagInput(event)">
        </div>
```

```
          <input type="hidden" id="tagsValues" name="tagsValues" value="">
          <div class="help-text">Enter tags to categorize your task</div>
        </div>

        <div class="form-actions">
          <button type="button" class="btn btn-secondary"
onclick="cancelForm()">Cancel</button>
          <button type="submit" class="btn btn-primary">Create in
Asana</button>
        </div>
      </form>
    </div>
  </div>
  <script>
    // Global variables
    let tags = [];
    let asanaProjects = [];
    let asanaSections = {};

    // Initialize the form
    function initialize() {
      // Set up form submission
      document.getElementById('asanaTaskForm').addEventListener('submit',
submitForm);

      // Set default due date to 7 days from now
      const dueDate = new Date();
      dueDate.setDate(dueDate.getDate() + 7);
      document.getElementById('dueDate').value =
dueDate.toISOString().split('T')[0];

      // Check Asana configuration
      checkAsanaConfig();
```

```
  }

  // Check Asana configuration
  function checkAsanaConfig() {
    google.script.run
      .withSuccessHandler(function(result) {
        if (result.configured) {
          // Load Asana data
          loadAsanaProjects();
          loadAsanaUsers();
        } else {
          // Show error message
          document.getElementById('errorMessage').textContent = 'Asana API is not configured. Please set up Asana API credentials in Settings.';
          document.getElementById('errorMessage').style.display = 'block';

          // Disable form
          document.querySelectorAll('input, select, textarea, button').forEach(function(element) {
            element.disabled = true;
          });
        }
      })
      .withFailureHandler(function(error) {
        document.getElementById('errorMessage').textContent = 'Error checking Asana configuration: ' + error;
        document.getElementById('errorMessage').style.display = 'block';
      })
      .checkAsanaConfig();
  }

  // Load Asana projects
  function loadAsanaProjects() {
```

```
    google.script.run
      .withSuccessHandler(function(projects) {
        asanaProjects = projects;
        const select = document.getElementById('projectSelect');

        // Clear loading option
        select.innerHTML = '<option value="">-- Select Project --</option>';

        // Add projects
        projects.forEach(function(project) {
          const option = document.createElement('option');
          option.value = project.gid;
          option.textContent = project.name;
          select.appendChild(option);
        });

        // Add project change handler
        select.addEventListener('change', handleProjectChange);
      })
      .withFailureHandler(function(error) {
        document.getElementById('errorMessage').textContent = 'Error loading
Asana projects: ' + error;
        document.getElementById('errorMessage').style.display = 'block';
      })
      .getAsanaProjects();
  }

  // Load Asana users
  function loadAsanaUsers() {
    google.script.run
      .withSuccessHandler(function(users) {
        const select = document.getElementById('assigneeSelect');
```

```
        // Clear loading option
        select.innerHTML = '<option value="">-- Unassigned --</option>';

        // Add users
        users.forEach(function(user) {
          const option = document.createElement('option');
          option.value = user.gid;
          option.textContent = user.name;
          select.appendChild(option);
        });
      })
      .withFailureHandler(function(error) {
        document.getElementById('errorMessage').textContent = 'Error loading
Asana users: ' + error;
        document.getElementById('errorMessage').style.display = 'block';
      })
      .getAsanaUsers();
  }

  // Handle project change
  function handleProjectChange() {
    const projectId = document.getElementById('projectSelect').value;
    const sectionSelect = document.getElementById('sectionSelect');

    // Reset sections
    sectionSelect.innerHTML = '<option value="">-- Loading Sections
--</option>';

    if (!projectId) {
      sectionSelect.innerHTML = '<option value="">-- Select Project First
--</option>';
      return;
    }
```

```
    // Check if we already have sections for this project
    if (asanaSections[projectId]) {
      populateSections(asanaSections[projectId]);
      return;
    }

    // Load sections
    google.script.run
      .withSuccessHandler(function(sections) {
        asanaSections[projectId] = sections;
        populateSections(sections);
      })
      .withFailureHandler(function(error) {
        document.getElementById('errorMessage').textContent = 'Error loading
project sections: ' + error;
        document.getElementById('errorMessage').style.display = 'block';
        sectionSelect.innerHTML = '<option value="">-- No Sections Found
--</option>';
      })
      .getAsanaProjectSections(projectId);
  }

  // Populate sections dropdown
  function populateSections(sections) {
    const select = document.getElementById('sectionSelect');

    // Clear loading option
    select.innerHTML = '<option value="">-- No Section --</option>';

    // Add sections
    sections.forEach(function(section) {
      const option = document.createElement('option');
```

```
      option.value = section.gid;
      option.textContent = section.name;
      select.appendChild(option);
    });
  }

  // Handle tag input
  function handleTagInput(event) {
    if (event.key === 'Enter') {
      event.preventDefault();

      const input = document.getElementById('tagsInput');
      const tag = input.value.trim();

      if (tag && !tags.includes(tag)) {
        // Add tag
        tags.push(tag);
        updateTagsDisplay();
        updateTagsValue();
      }

      input.value = '';
    }
  }

  // Remove tag
  function removeTag(tag) {
    const index = tags.indexOf(tag);
    if (index !== -1) {
      tags.splice(index, 1);
      updateTagsDisplay();
      updateTagsValue();
    }
```

```
  }

  // Update tags display
  function updateTagsDisplay() {
    const container = document.getElementById('tagsContainer');

    // Remove existing tags
    document.querySelectorAll('.tag').forEach(function(element) {
      element.remove();
    });

    // Add tags
    const input = document.getElementById('tagsInput');
    tags.forEach(function(tag) {
      const tagElement = document.createElement('div');
      tagElement.className = 'tag';
      tagElement.innerHTML = `${tag} <span class="tag-remove"
onclick="removeTag('${tag}')">&times;</span>`;

      container.insertBefore(tagElement, input);
    });
  }

  // Update hidden tags value
  function updateTagsValue() {
    document.getElementById('tagsValues').value = JSON.stringify(tags);
  }

  // Submit form
  function submitForm(event) {
    event.preventDefault();

    // Hide messages
```

```
    document.getElementById('successMessage').style.display = 'none';
    document.getElementById('errorMessage').style.display = 'none';

    // Validate required fields
    const taskName = document.getElementById('taskName').value;
    const projectId = document.getElementById('projectSelect').value;

    if (!taskName) {
      document.getElementById('errorMessage').textContent = 'Please enter a
task name.';
      document.getElementById('errorMessage').style.display = 'block';
      return;
    }

    if (!projectId) {
      document.getElementById('errorMessage').textContent = 'Please select a
project.';
      document.getElementById('errorMessage').style.display = 'block';
      return;
    }

    // Get form data
    const taskData = {
      name: taskName,
      notes: document.getElementById('taskNotes').value,
      projectId: projectId,
      assigneeId: document.getElementById('assigneeSelect').value,
      dueDate: document.getElementById('dueDate').value,
      sectionId: document.getElementById('sectionSelect').value,
      tags: tags
    };

    // Submit to server
```

```
    google.script.run
      .withSuccessHandler(function(result) {
        if (result.success) {
          // Show success message
          document.getElementById('successMessage').textContent =
result.message || 'Task created successfully in Asana!';
          document.getElementById('successMessage').style.display = 'block';

          // Reset form
          document.getElementById('asanaTaskForm').reset();
          tags = [];
          updateTagsDisplay();

          // Set default due date again
          const dueDate = new Date();
          dueDate.setDate(dueDate.getDate() + 7);
          document.getElementById('dueDate').value =
dueDate.toISOString().split('T')[0];
        } else {
          // Show error message
          document.getElementById('errorMessage').textContent =
result.message || 'An error occurred while creating the task in Asana.';
          document.getElementById('errorMessage').style.display = 'block';
        }
      })
      .withFailureHandler(function(error) {
        // Show error message
        document.getElementById('errorMessage').textContent = 'Error creating
Asana task: ' + error;
        document.getElementById('errorMessage').style.display = 'block';
      })
      .createAsanaTask(taskData);
  }
```

```
    // Cancel form
    function cancelForm() {
      google.script.host.close();
    }

    // Initialize when the page loads
    google.script.onload = function() {
      initialize();
    };
  </script>
</body>
</html>
<!DOCTYPE html>
<html>
<head>
  <base target="_top">
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link
href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500;700&fam
ily=Poppins:wght@500;600;700&display=swap" rel="stylesheet">
  <style>
    /* Variables for Trevor Project color scheme */
    :root {
      /* Primary Colors */
      --trevor-orange: #FF786E;
      --deep-blue: #001A4E;
      --purple: #9A3499;
      --teal: #137F6A;

      /* Secondary Colors */
      --light-blue: #4F52DE;
```

```
    --soft-yellow: #FFAD8D;
    --lavender: #B3AE4A;
    --light-purple: #F54AC;

    /* Tertiary Colors */
    --soft-green: #BAE2CE;
    --pale-yellow: #FFF2DF;
    --light-pink: #FBCBBE;
    --soft-lavender: #D1CFCC;

    /* Gradients */
    --primary-gradient: linear-gradient(135deg, var(--trevor-orange),
var(--light-pink));
    --calm-gradient: linear-gradient(135deg, var(--light-blue),
var(--soft-green));
    --support-gradient: linear-gradient(45deg, var(--purple),
var(--light-purple));
    --background-gradient: linear-gradient(180deg, var(--pale-yellow),
#FFFFFF);

    /* Spacing */
    --spacing-xs: 4px;
    --spacing-sm: 8px;
    --spacing-md: 16px;
    --spacing-lg: 24px;
    --spacing-xl: 32px;

    /* Borders */
    --border-radius-sm: 8px;
    --border-radius-md: 12px;
    --border-radius-lg: 16px;
  }
```

```css
body {
  font-family: 'Roboto', sans-serif;
  margin: 0;
  padding: 0;
  background-color: var(--pale-yellow);
  color: var(--deep-blue);
}

.container {
  max-width: 900px;
  margin: 0 auto;
  padding: var(--spacing-md);
}

h1, h2, h3, h4 {
  font-family: 'Poppins', sans-serif;
  color: var(--deep-blue);
}

.header {
  margin-bottom: var(--spacing-lg);
  text-align: center;
}

.card {
  background: white;
  border-radius: var(--border-radius-md);
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.05);
  padding: var(--spacing-lg);
  margin-bottom: var(--spacing-lg);
  position: relative;
}
```

```css
.card::before {
  content: '';
  position: absolute;
  left: 0;
  top: 0;
  height: 100%;
  width: 3px;
  background: var(--trevor-orange);
}

.priority-card::before {
  background: var(--light-purple);
}

.form-group {
  margin-bottom: var(--spacing-md);
}

label {
  display: block;
  font-weight: 500;
  margin-bottom: var(--spacing-xs);
}

input, select, textarea {
  width: 100%;
  padding: var(--spacing-sm);
  border: 1px solid #ddd;
  border-radius: var(--border-radius-sm);
  font-family: 'Roboto', sans-serif;
  font-size: 16px;
}
```

```css
textarea {
  resize: vertical;
  min-height: 80px;
}

.btn {
  border: none;
  padding: var(--spacing-md) var(--spacing-lg);
  border-radius: var(--border-radius-sm);
  font-weight: 500;
  cursor: pointer;
  transition: all 0.2s ease;
}

.btn-primary {
  background: var(--primary-gradient);
  color: white;
}

.btn-primary:hover {
  box-shadow: 0 4px 8px rgba(255, 120, 110, 0.3);
  transform: translateY(-2px);
}

.btn-secondary {
  background: white;
  color: var(--deep-blue);
  border: 1px solid #ddd;
}

.btn-secondary:hover {
  background: #f5f5f5;
}
```

```css
.btn-priority {
  background: linear-gradient(45deg, var(--purple), var(--light-purple));
  color: white;
}

.btn-priority:hover {
  box-shadow: 0 4px 8px rgba(155, 52, 153, 0.3);
  transform: translateY(-2px);
}

.btn-sm {
  font-size: 14px;
  padding: var(--spacing-xs) var(--spacing-sm);
}

.center {
  text-align: center;
}

.section-actions {
  display: flex;
  justify-content: space-between;
  margin-top: var(--spacing-md);
}

.form-actions {
  display: flex;
  justify-content: space-between;
  margin-top: var(--spacing-lg);
}

.tab-container {
```

```
  margin-bottom: var(--spacing-lg);
}

.tabs {
  display: flex;
  margin-bottom: var(--spacing-md);
  border-bottom: 1px solid #ddd;
}

.tab {
  padding: var(--spacing-sm) var(--spacing-md);
  cursor: pointer;
  border-bottom: 3px solid transparent;
  transition: all 0.2s ease;
}

.tab.active {
  border-bottom-color: var(--trevor-orange);
  font-weight: 500;
}

.tab.priority.active {
  border-bottom-color: var(--light-purple);
}

.tab-content {
  display: none;
}

.tab-content.active {
  display: block;
}
```

```css
.success-message {
  background-color: var(--soft-green);
  border-radius: var(--border-radius-sm);
  padding: var(--spacing-md);
  margin-bottom: var(--spacing-md);
  display: none;
}

.error-message {
  background-color: var(--light-pink);
  border-radius: var(--border-radius-sm);
  padding: var(--spacing-md);
  margin-bottom: var(--spacing-md);
  display: none;
}

.counselor-selector {
  padding: var(--spacing-md);
  margin-bottom: var(--spacing-lg);
  background: white;
  border-radius: var(--border-radius-md);
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.05);
}

.coaching-type {
  margin-bottom: var(--spacing-md);
}

.coaching-type-selector {
  display: flex;
  gap: var(--spacing-md);
  margin-bottom: var(--spacing-md);
}
```

```css
.coaching-type-option {
  flex: 1;
  padding: var(--spacing-md);
  border: 2px solid #ddd;
  border-radius: var(--border-radius-md);
  text-align: center;
  cursor: pointer;
  transition: all 0.2s ease;
}

.coaching-type-option:hover {
  border-color: var(--trevor-orange);
  background-color: rgba(255, 120, 110, 0.05);
}

.coaching-type-option.selected {
  border-color: var(--trevor-orange);
  background-color: rgba(255, 120, 110, 0.1);
}

.coaching-type-option.priority {
  border-color: #ddd;
}

.coaching-type-option.priority:hover {
  border-color: var(--light-purple);
  background-color: rgba(155, 52, 153, 0.05);
}

.coaching-type-option.priority.selected {
  border-color: var(--light-purple);
  background-color: rgba(155, 52, 153, 0.1);
```

```css
}

.coaching-type-icon {
  font-size: 24px;
  margin-bottom: var(--spacing-xs);
}

.coaching-type-label {
  font-weight: 500;
}

.coaching-type-description {
  font-size: 12px;
  color: #666;
  margin-top: var(--spacing-xs);
}

.previous-notes {
  margin-top: var(--spacing-lg);
}

.note-item {
  padding: var(--spacing-sm);
  margin-bottom: var(--spacing-sm);
  border-bottom: 1px solid #eee;
  cursor: pointer;
}

.note-item:hover {
  background: #f9f9f9;
}

.note-date {
```

```
  font-weight: 500;
}

.note-summary {
  font-size: 0.9em;
  color: #666;
  margin-top: var(--spacing-xs);
}

.note-type {
  display: inline-block;
  padding: 2px 8px;
  border-radius: 12px;
  font-size: 12px;
  background-color: var(--soft-green);
  color: var(--deep-blue);
  margin-left: var(--spacing-xs);
}

.note-type.priority {
  background-color: var(--light-purple);
  color: white;
}

/* Counselor details card */
.counselor-details-card {
  background: white;
  border-radius: var(--border-radius-md);
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.05);
  padding: var(--spacing-md);
  margin-bottom: var(--spacing-lg);
  display: flex;
  flex-direction: column;
```

```css
}

.counselor-details-header {
  display: flex;
  align-items: center;
  margin-bottom: var(--spacing-md);
}

.counselor-avatar {
  width: 60px;
  height: 60px;
  border-radius: 50%;
  background: var(--primary-gradient);
  color: white;
  display: flex;
  align-items: center;
  justify-content: center;
  font-size: 24px;
  font-weight: bold;
  margin-right: var(--spacing-md);
}

.counselor-avatar.priority {
  background: var(--support-gradient);
}

.counselor-info {
  flex: 1;
}

.counselor-name {
  font-size: 20px;
  font-weight: 600;
```

```
  margin-bottom: 4px;
}

.counselor-status {
  display: inline-block;
  padding: 4px 8px;
  border-radius: 12px;
  font-size: 12px;
  font-weight: 500;
  text-transform: uppercase;
}

.status-active {
  background-color: var(--soft-green);
  color: var(--deep-blue);
}

.status-inactive {
  background-color: var(--light-purple);
  color: white;
}

.status-leave {
  background-color: var(--soft-yellow);
  color: var(--deep-blue);
}

.counselor-metadata {
  display: flex;
  flex-wrap: wrap;
  gap: var(--spacing-md);
}
```

```css
.metadata-item {
  flex: 1;
  min-width: 150px;
}

.metadata-label {
  font-size: 12px;
  color: #666;
  margin-bottom: 2px;
}

.metadata-value {
  font-weight: 500;
}

.section-header {
  margin-bottom: var(--spacing-md);
  padding-bottom: var(--spacing-xs);
  border-bottom: 1px solid #eee;
}

.loading-spinner {
  display: inline-block;
  width: 20px;
  height: 20px;
  border: 3px solid rgba(255, 120, 110, 0.3);
  border-radius: 50%;
  border-top-color: var(--trevor-orange);
  animation: spin 1s ease-in-out infinite;
  margin-right: 8px;
}

@keyframes spin {
```

```
    to { transform: rotate(360deg); }
  }

  .loading-text {
    display: flex;
    align-items: center;
    justify-content: center;
    height: 100px;
    color: #666;
  }
 </style>
</head>
<body>
 <div class="container">
  <div class="header">
    <h1>Coaching Notes</h1>
    <p>Document regular or priority coaching interactions</p>
  </div>

  <div id="successMessage" class="success-message">
    Coaching note saved successfully!
  </div>

  <div id="errorMessage" class="error-message">
    An error occurred while saving the coaching note. Please try again.
  </div>

  <div class="counselor-selector">
    <div class="form-group">
      <label for="counselorSelect">Select Counselor</label>
      <select id="counselorSelect" onchange="counselorSelected()">
        <option value="">-- Select Counselor --</option>
        <!-- Counselors will be populated here -->
```

```
      </select>
    </div>
  </div>

  <!-- Counselor Details Card -->
  <div id="counselorDetailsCard" class="counselor-details-card"
style="display: none;">
    <div class="counselor-details-header">
      <div id="counselorAvatar" class="counselor-avatar">
        <!-- First letter of name will be here -->
      </div>
      <div class="counselor-info">
        <div id="counselorNameDisplay" class="counselor-name"></div>
        <span id="counselorStatusBadge" class="counselor-status"></span>
      </div>
    </div>

    <div class="counselor-metadata">
      <div class="metadata-item">
        <div class="metadata-label">Email</div>
        <div id="counselorEmailDisplay" class="metadata-value"></div>
      </div>
      <div class="metadata-item">
        <div class="metadata-label">Role</div>
        <div id="counselorRoleDisplay" class="metadata-value"></div>
      </div>
      <div class="metadata-item">
        <div class="metadata-label">Start Date</div>
        <div id="counselorStartDateDisplay" class="metadata-value"></div>
      </div>
    </div>
  </div>
```

```
   <div id="coachingContent" style="display: none;">
     <div class="coaching-type">
       <h3>Select Coaching Type</h3>
       <div class="coaching-type-selector">
         <div class="coaching-type-option"
onclick="selectCoachingType('regular')">
           <div class="coaching-type-icon">📝</div>
           <div class="coaching-type-label">Regular Coaching</div>
           <div class="coaching-type-description">Standard coaching
interaction</div>
         </div>
         <div class="coaching-type-option priority"
onclick="selectCoachingType('priority')">
           <div class="coaching-type-icon">🚨</div>
           <div class="coaching-type-label">Priority Coaching</div>
           <div class="coaching-type-description">Performance concerns
requiring immediate attention</div>
         </div>
       </div>
     </div>

     <div class="tab-container">
       <div class="tabs" id="tabs">
         <div class="tab active" onclick="showTab('infoTab')">Basic Info</div>
         <div class="tab" id="preparationTab"
onclick="showTab('preparationTab')">Preparation</div>
         <div class="tab" id="interventionTab"
onclick="showTab('interventionTab')">Intervention</div>
         <div class="tab" id="followUpTab"
onclick="showTab('followUpTab')">Follow-Up</div>
       </div>

       <!-- Basic Info Tab -->
```

```
      <div id="infoTab" class="tab-content active">
        <div class="card" id="infoCard">
          <h3>Coaching Basic Information</h3>

          <div class="form-group">
            <label for="coachingDate">Coaching Date</label>
            <input type="date" id="coachingDate" name="coachingDate"
required>
          </div>

          <div class="form-group">
            <label for="coachingType">Coaching Type</label>
            <select id="coachingType" name="coachingType" required>
              <option value="Regular">Regular Coaching</option>
              <option value="Priority">Priority Coaching</option>
            </select>
          </div>

          <div class="form-group">
            <label for="coachFocus">Coaching Focus</label>
            <input type="text" id="coachFocus" name="coachFocus"
placeholder="Enter the main focus of this coaching session" required>
          </div>

          <div class="form-group">
            <label for="coachSummary">Brief Summary</label>
            <textarea id="coachSummary" name="coachSummary"
placeholder="Brief description of the coaching interaction"
required></textarea>
          </div>

          <div class="form-group">
            <label for="followUpDate">Follow-Up Date</label>
```

```
            <input type="date" id="followUpDate" name="followUpDate">
          </div>

          <div class="section-actions">
            <button type="button" class="btn btn-secondary"
onclick="google.script.host.close()">Cancel</button>
            <button type="button" class="btn btn-primary"
id="nextToPreparationBtn" onclick="showTab('preparationTab')">Next:
Preparation</button>
          </div>
        </div>
      </div>

      <!-- Preparation Tab -->
      <div id="preparationTab" class="tab-content">
        <div class="card" id="preparationCard">
          <h3>Coaching Preparation</h3>

          <div class="form-group">
            <label>Initial Assessment</label>
            <div>
              <input type="checkbox" id="prepGatherConcerns">
              <label for="prepGatherConcerns">Gather specific
performance/behavioral concerns</label>
            </div>
            <div>
              <input type="checkbox" id="prepReviewDocumentation">
              <label for="prepReviewDocumentation">Review relevant
documentation</label>
            </div>
            <div>
              <input type="checkbox" id="prepCollectEvidence">
```

```
              <label for="prepCollectEvidence">Collect objective
evidence</label>
            </div>
            <div>
              <input type="checkbox" id="prepIdentifyRootCauses">
              <label for="prepIdentifyRootCauses">Identify potential root
causes</label>
            </div>
          </div>

          <div class="form-group">
            <label for="concernAreas">Documented Concern Areas</label>
            <textarea id="concernAreas" name="concernAreas"
placeholder="Document specific instances of concern, deviation from expected
standards, impact on team/service quality, frequency and severity of
issues"></textarea>
          </div>

          <div class="form-group">
            <label>Preparation Checklist</label>
            <div>
              <input type="checkbox" id="prepVerifyDocumentation">
              <label for="prepVerifyDocumentation">Verify all documentation
is accurate</label>
            </div>
            <div>
              <input type="checkbox" id="prepPrepareExamples">
              <label for="prepPrepareExamples">Prepare specific, observable
examples</label>
            </div>
            <div>
              <input type="checkbox" id="prepIdentifyResources">
```

```
              <label for="prepIdentifyResources">Identify potential support
resources</label>
            </div>
            <div>
              <input type="checkbox" id="prepDevelopApproach">
              <label for="prepDevelopApproach">Develop clear, constructive
approach</label>
            </div>
          </div>

          <div class="form-group">
            <label for="preparationReflection">Pre-Coaching
Reflection</label>
            <textarea id="preparationReflection" name="preparationReflection"
placeholder="What is the primary goal of this coaching intervention? What
specific behaviors need to change? What support can be provided? What are the
potential barriers to improvement?"></textarea>
          </div>

          <div class="section-actions">
            <button type="button" class="btn btn-secondary"
onclick="showTab('infoTab')">Back: Basic Info</button>
            <button type="button" class="btn btn-primary"
id="nextToInterventionBtn" onclick="showTab('interventionTab')">Next:
Intervention</button>
          </div>
        </div>
      </div>

      <!-- Intervention Tab -->
      <div id="interventionTab" class="tab-content">
        <div class="card" id="interventionCard">
          <h3>Coaching Intervention</h3>
```

```
            <div class="form-group">
              <label for="initialConnection">Initial Connection</label>
              <textarea id="initialConnection" name="initialConnection"
placeholder="Document how you established psychological safety, created a
collaborative approach, and clarified the purpose of the meeting"></textarea>
            </div>

            <div class="form-group">
              <label for="concernExploration">Concern Exploration</label>
              <textarea id="concernExploration" name="concernExploration"
placeholder="Document objective observations using factual language, impact
assessment on performance/team/quality, and root cause
exploration"></textarea>
            </div>

            <div class="form-group">
              <label for="developmentPlanning">Development Planning</label>
              <textarea id="developmentPlanning" name="developmentPlanning"
placeholder="Document immediate corrective actions, skill development pathway,
and support framework"></textarea>
            </div>

            <div class="section-actions">
              <button type="button" class="btn btn-secondary"
onclick="showTab('preparationTab')">Back: Preparation</button>
              <button type="button" class="btn btn-primary"
id="nextToFollowUpBtn" onclick="showTab('followUpTab')">Next:
Follow-Up</button>
            </div>
          </div>
        </div>
```

```
      <!-- Follow-Up Tab -->
      <div id="followUpTab" class="tab-content">
        <div class="card" id="followUpCard">
          <h3>Post-Coaching Documentation</h3>

          <div class="form-group">
            <label>Immediate Documentation</label>
            <div>
              <input type="checkbox" id="docSummarizePoints">
              <label for="docSummarizePoints">Summarize key discussion
points</label>
            </div>
            <div>
              <input type="checkbox" id="docAgreedActions">
              <label for="docAgreedActions">Document agreed-upon
actions</label>
            </div>
            <div>
              <input type="checkbox" id="docClarifyExpectations">
              <label for="docClarifyExpectations">Clarify
expectations</label>
            </div>
            <div>
              <input type="checkbox" id="docEstablishTimeline">
              <label for="docEstablishTimeline">Establish follow-up
timeline</label>
            </div>
          </div>

          <div class="form-group">
            <label for="formalIntervention">Formal Coaching Intervention
Record</label>
```

```
            <textarea id="formalIntervention" name="formalIntervention"
placeholder="Document date of coaching session, primary concerns addressed,
agreed-upon action steps, support resources provided, and follow-up
schedule"></textarea>
          </div>

          <div class="form-group">
            <label for="actionItems">Action Item Tracking</label>
            <textarea id="actionItems" name="actionItems"
placeholder="Document specific actions, owners, target completion dates,
required resources, current status, and verification methods"></textarea>
          </div>

          <div class="form-group">
            <label for="confidentialNotes">Confidential Development
Notes</label>
            <textarea id="confidentialNotes" name="confidentialNotes"
placeholder="Document discrete observations, potential long-term development
areas, strengths to leverage, and potential growth opportunities"></textarea>
          </div>

          <div class="section-actions">
            <button type="button" class="btn btn-secondary"
onclick="showTab('interventionTab')">Back: Intervention</button>
            <button type="button" class="btn btn-primary" id="saveButton"
onclick="saveCoachingNote()">Save Coaching Note</button>
          </div>
        </div>
      </div>
    </div>
  </div>
```

```
    <div id="previousNotesSection" style="display: none;" class="card
previous-notes">
      <h3>Previous Coaching Notes</h3>
      <div id="previousNotesList">
        <div class="loading-text">
          <div class="loading-spinner"></div>
          Loading previous notes...
        </div>
      </div>
    </div>
  </div>
   <script>
    // Global variables
    let isPriorityCoaching = false;

    // Initialize date field with today's date
    document.addEventListener('DOMContentLoaded', function() {
      const today = new Date().toISOString().split('T')[0];
      document.getElementById('coachingDate').value = today;

      // Calculate a default follow-up date (one week from today)
      const followUp = new Date();
      followUp.setDate(followUp.getDate() + 7);
      document.getElementById('followUpDate').value =
followUp.toISOString().split('T')[0];

      // Load counselors from Counselor Tracking sheet
      loadCounselors();
    });

    // Load counselors from the server
    function loadCounselors() {
      google.script.run
```

```
    .withSuccessHandler(populateCounselors)
    .withFailureHandler(handleError)
    .getAllCounselors();
}

// Populate counselor dropdown
function populateCounselors(counselors) {
  const select = document.getElementById('counselorSelect');

  // Clear existing options (except the first one)
  while (select.options.length > 1) {
    select.remove(1);
  }

  // Filter for valid counselors and sort alphabetically by name
  const validCounselors = counselors.filter(counselor =>
    counselor.name && counselor.email
  );

  validCounselors.sort((a, b) => a.name.localeCompare(b.name));

  // Add counselors to dropdown
  if (validCounselors.length > 0) {
    validCounselors.forEach(counselor => {
      const option = document.createElement('option');
      option.value = counselor.email;
      option.text = counselor.name;
      option.dataset.status = counselor.status || 'active';
      option.dataset.role = counselor.role || '';
      option.dataset.startDate = counselor.startDate || '';

      // Add status indicator if not active
```

```
        if (counselor.status && counselor.status.toLowerCase() !== 'active')
{
          option.text += ` (${counselor.status})`;
        }

        select.add(option);
      });
    } else {
      // Add a placeholder option if no counselors
      const option = document.createElement('option');
      option.value = '';
      option.text = 'No counselors found';
      option.disabled = true;
      select.add(option);
    }
  }

  // Handle counselor selection
  function counselorSelected() {
    const select = document.getElementById('counselorSelect');
    const selectedOption = select.options[select.selectedIndex];
    const counselorEmail = select.value;

    // Clear any existing messages
    document.getElementById('successMessage').style.display = 'none';
    document.getElementById('errorMessage').style.display = 'none';

    if (counselorEmail) {
      // Show coaching content section
      document.getElementById('coachingContent').style.display = 'block';

      // Display counselor details
      displayCounselorDetails({
```

```
        name: selectedOption.text.split(' (')[0], // Remove status if present
        email: counselorEmail,
        status: selectedOption.dataset.status || 'active',
        role: selectedOption.dataset.role || 'Counselor',
        startDate: selectedOption.dataset.startDate || ''
      });

      // Load previous coaching notes for this counselor
      loadPreviousNotes(counselorEmail);
    } else {
      // Hide sections if no counselor selected
      document.getElementById('coachingContent').style.display = 'none';
      document.getElementById('counselorDetailsCard').style.display = 'none';
      document.getElementById('previousNotesSection').style.display = 'none';
    }
  }

  // Display counselor details
  function displayCounselorDetails(counselor) {
    // Show counselor details card
    const detailsCard = document.getElementById('counselorDetailsCard');
    detailsCard.style.display = 'block';

    // Avatar (first letter of name)
    const avatar = document.getElementById('counselorAvatar');
    avatar.textContent = counselor.name ?
counselor.name.charAt(0).toUpperCase() : '?';
    avatar.className = 'counselor-avatar'; // Reset class

    // Name
    document.getElementById('counselorNameDisplay').textContent =
counselor.name;
```

```
    // Status Badge
    const statusBadge = document.getElementById('counselorStatusBadge');
    statusBadge.textContent = counselor.status;
    statusBadge.className = `counselor-status
status-${counselor.status.toLowerCase()}`;

    // Email
    document.getElementById('counselorEmailDisplay').textContent =
counselor.email;

    // Role
    document.getElementById('counselorRoleDisplay').textContent =
counselor.role || 'Counselor';

    // Start Date
    document.getElementById('counselorStartDateDisplay').textContent =
      counselor.startDate ? new
Date(counselor.startDate).toLocaleDateString() : 'Not available';
  }

  // Select coaching type
  function selectCoachingType(type) {
    // Update UI
    const regularOption =
document.querySelector('.coaching-type-option:not(.priority)');
    const priorityOption =
document.querySelector('.coaching-type-option.priority');

    regularOption.classList.remove('selected');
    priorityOption.classList.remove('selected');

    if (type === 'priority') {
      priorityOption.classList.add('selected');
```

```
        document.getElementById('coachingType').value = 'Priority';
        isPriorityCoaching = true;

        // Update counselor avatar
        document.getElementById('counselorAvatar').classList.add('priority');

        // Update styling for priority coaching
        document.querySelectorAll('.card').forEach(card => {
          card.classList.add('priority-card');
        });

        document.querySelectorAll('.tab').forEach(tab => {
          tab.classList.add('priority');
        });

        document.querySelectorAll('.btn-primary').forEach(btn => {
          btn.classList.remove('btn-primary');
          btn.classList.add('btn-priority');
        });
      } else {
        regularOption.classList.add('selected');
        document.getElementById('coachingType').value = 'Regular';
        isPriorityCoaching = false;

        // Remove priority class from counselor avatar
document.getElementById('counselorAvatar').classList.remove('priority');

        // Revert styling
        document.querySelectorAll('.card').forEach(card => {
          card.classList.remove('priority-card');
        });
```

```
      document.querySelectorAll('.tab').forEach(tab => {
        tab.classList.remove('priority');
      });

      document.querySelectorAll('.btn-priority').forEach(btn => {
        btn.classList.remove('btn-priority');
        btn.classList.add('btn-primary');
      });
    }
  }

  // Show the selected tab
  function showTab(tabId) {
    // Hide all tabs
    document.querySelectorAll('.tab-content').forEach(tab => {
      tab.classList.remove('active');
    });

    // Show the selected tab
    document.getElementById(tabId).classList.add('active');

    // Update tab buttons
    document.querySelectorAll('.tab').forEach(tab => {
      tab.classList.remove('active');
    });

    // Highlight the active tab button
    const tabButtons = document.querySelectorAll('.tab');
    const tabIds = ['infoTab', 'preparationTab', 'interventionTab',
'followUpTab'];
    const tabIndex = tabIds.indexOf(tabId);

    if (tabIndex >= 0 && tabIndex < tabButtons.length) {
```

```
      tabButtons[tabIndex].classList.add('active');
    }
  }

  // Load previous coaching notes for the selected counselor
  function loadPreviousNotes(counselorEmail) {
    const section = document.getElementById('previousNotesSection');
    section.style.display = 'block';

    const container = document.getElementById('previousNotesList');
    container.innerHTML = '<div class="loading-text"><div
class="loading-spinner"></div>Loading previous notes...</div>';

    google.script.run
      .withSuccessHandler(displayPreviousNotes)
      .withFailureHandler(function(error) {
        console.error('Error loading previous notes:', error);
        container.innerHTML = '<p class="center">Error loading previous
coaching notes. Please try again.</p>';
      })
      .getCoachingNotes(counselorEmail);
  }

  // Display previous coaching notes
  function displayPreviousNotes(notes) {
    const container = document.getElementById('previousNotesList');

    // Clear loading indicator
    container.innerHTML = '';

    // Show or hide section based on notes
    if (notes && notes.length > 0) {
      // Sort notes by date (newest first)
```

```
      notes.sort((a, b) => new Date(b.date) - new Date(a.date));

      // Add notes to the list
      notes.forEach(note => {
        const noteEl = document.createElement('div');
        noteEl.className = 'note-item';
        noteEl.onclick = function() { loadNoteDetails(note.id); };

        const dateEl = document.createElement('div');
        dateEl.className = 'note-date';
        dateEl.textContent = new Date(note.date).toLocaleDateString();

        const typeEl = document.createElement('span');
        typeEl.className = note.type === 'Priority' ? 'note-type priority' :
'note-type';
        typeEl.textContent = note.type;
        dateEl.appendChild(typeEl);

        const summaryEl = document.createElement('div');
        summaryEl.className = 'note-summary';
        summaryEl.textContent = note.summary || 'No summary available';

        noteEl.appendChild(dateEl);
        noteEl.appendChild(summaryEl);
        container.appendChild(noteEl);
      });
    } else {
      // Show message if no notes
      container.innerHTML = '<p class="center">No previous coaching notes
found.</p>';
    }
  }
```

```
  // Load coaching note details
  function loadNoteDetails(noteId) {
    // Show loading state in previous notes list
    document.getElementById('previousNotesList').innerHTML =
      '<div class="loading-text"><div class="loading-spinner"></div>Loading
note details...</div>';

    google.script.run
      .withSuccessHandler(populateNoteDetails)
      .withFailureHandler(function(error) {
        console.error('Error loading note details:', error);
        document.getElementById('errorMessage').style.display = 'block';
        document.getElementById('errorMessage').textContent = 'Could not load
coaching note details.';
        // Reload the notes list
        const counselorEmail =
document.getElementById('counselorSelect').value;
        loadPreviousNotes(counselorEmail);
      })
      .getCoachingNoteDetails(noteId);
  }

  // Populate coaching note details into the form
  function populateNoteDetails(note) {
    if (!note) {
      document.getElementById('errorMessage').style.display = 'block';
      document.getElementById('errorMessage').textContent = 'Note details not
found.';
      const counselorEmail =
document.getElementById('counselorSelect').value;
      loadPreviousNotes(counselorEmail);
      return;
    }
```

```
    // Set coaching type
    selectCoachingType(note.type === 'Priority' ? 'priority' : 'regular');

    // Set basic info
    document.getElementById('coachingDate').value = formatDate(note.date);
    document.getElementById('coachFocus').value = note.focus || '';
    document.getElementById('coachSummary').value = note.summary || '';
    document.getElementById('followUpDate').value =
formatDate(note.followUpDate);

    // Set preparation tab data
    document.getElementById('prepGatherConcerns').checked =
note.prepGatherConcerns || false;
    document.getElementById('prepReviewDocumentation').checked =
note.prepReviewDocumentation || false;
    document.getElementById('prepCollectEvidence').checked =
note.prepCollectEvidence || false;
    document.getElementById('prepIdentifyRootCauses').checked =
note.prepIdentifyRootCauses || false;
    document.getElementById('concernAreas').value = note.concernAreas || '';
    document.getElementById('prepVerifyDocumentation').checked =
note.prepVerifyDocumentation || false;
    document.getElementById('prepPrepareExamples').checked =
note.prepPrepareExamples || false;
    document.getElementById('prepIdentifyResources').checked =
note.prepIdentifyResources || false;
    document.getElementById('prepDevelopApproach').checked =
note.prepDevelopApproach || false;
    document.getElementById('preparationReflection').value =
note.preparationReflection || '';

    // Set intervention tab data
```

```
    document.getElementById('initialConnection').value =
note.initialConnection || '';
    document.getElementById('concernExploration').value =
note.concernExploration || '';
    document.getElementById('developmentPlanning').value =
note.developmentPlanning || '';

    // Set follow-up tab data
    document.getElementById('docSummarizePoints').checked =
note.docSummarizePoints || false;
    document.getElementById('docAgreedActions').checked =
note.docAgreedActions || false;
    document.getElementById('docClarifyExpectations').checked =
note.docClarifyExpectations || false;
    document.getElementById('docEstablishTimeline').checked =
note.docEstablishTimeline || false;
    document.getElementById('formalIntervention').value =
note.formalIntervention || '';
    document.getElementById('actionItems').value = note.actionItems || '';
    document.getElementById('confidentialNotes').value =
note.confidentialNotes || '';

    // Show the first tab
    showTab('infoTab');

    // Reload the notes list to remove loading indicator
    const counselorEmail = document.getElementById('counselorSelect').value;
    loadPreviousNotes(counselorEmail);
  }

  // Format date for input field (YYYY-MM-DD)
  function formatDate(dateString) {
    if (!dateString) return '';
```

```
    const date = new Date(dateString);
    if (isNaN(date.getTime())) return '';
    return date.toISOString().split('T')[0];
  }

  // Save coaching note
  function saveCoachingNote() {
    // Get counselor data
    const counselorSelect = document.getElementById('counselorSelect');
    const selectedOption =
counselorSelect.options[counselorSelect.selectedIndex];

    if (!counselorSelect.value) {
      alert('Please select a counselor');
      return;
    }

    // Validate required fields
    const coachingDate = document.getElementById('coachingDate').value;
    const coachFocus = document.getElementById('coachFocus').value;
    const coachSummary = document.getElementById('coachSummary').value;

    if (!coachingDate || !coachFocus || !coachSummary) {
      alert('Please fill in all required fields (Coaching Date, Focus, and
Summary)');
      return;
    }

    // Prepare coaching note data
    const noteData = {
      counselorName: selectedOption.textContent.split(' (')[0], // Remove
status if present
      counselorEmail: counselorSelect.value,
```

```
        date: coachingDate,
        type: isPriorityCoaching ? 'Priority' : 'Regular',
        focus: coachFocus,
        summary: coachSummary,
        followUpDate: document.getElementById('followUpDate').value,

        // Preparation data
        prepGatherConcerns:
document.getElementById('prepGatherConcerns').checked,
        prepReviewDocumentation:
document.getElementById('prepReviewDocumentation').checked,
        prepCollectEvidence:
document.getElementById('prepCollectEvidence').checked,
        prepIdentifyRootCauses:
document.getElementById('prepIdentifyRootCauses').checked,
        concernAreas: document.getElementById('concernAreas').value,
        prepVerifyDocumentation:
document.getElementById('prepVerifyDocumentation').checked,
        prepPrepareExamples:
document.getElementById('prepPrepareExamples').checked,
        prepIdentifyResources:
document.getElementById('prepIdentifyResources').checked,
        prepDevelopApproach:
document.getElementById('prepDevelopApproach').checked,
        preparationReflection:
document.getElementById('preparationReflection').value,

        // Intervention data
        initialConnection: document.getElementById('initialConnection').value,
        concernExploration:
document.getElementById('concernExploration').value,
        developmentPlanning:
document.getElementById('developmentPlanning').value,
```

```
      // Follow-up data
      docSummarizePoints:
document.getElementById('docSummarizePoints').checked,
      docAgreedActions: document.getElementById('docAgreedActions').checked,
      docClarifyExpectations:
document.getElementById('docClarifyExpectations').checked,
      docEstablishTimeline:
document.getElementById('docEstablishTimeline').checked,
      formalIntervention:
document.getElementById('formalIntervention').value,
      actionItems: document.getElementById('actionItems').value,
      confidentialNotes: document.getElementById('confidentialNotes').value
    };

    // Show loading state
    const saveButton = document.getElementById('saveButton');
    const originalText = saveButton.textContent;
    saveButton.innerHTML = '<div class="loading-spinner"></div> Saving...';
    saveButton.disabled = true;

    // Clear previous messages
    document.getElementById('successMessage').style.display = 'none';
    document.getElementById('errorMessage').style.display = 'none';

    // Submit to the server
    google.script.run
      .withSuccessHandler(function(result) {
        // Reset button state
        saveButton.textContent = originalText;
        saveButton.disabled = false;

        if (result.success) {
```

```
            // Show success message
            document.getElementById('successMessage').style.display = 'block';
            document.getElementById('successMessage').textContent =
result.message || 'Coaching note saved successfully!';

            // Ask if user wants to send follow-up email
            const sendEmail = confirm('Coaching note saved successfully. Would
you like to send a follow-up email to the counselor?');

            if (sendEmail) {
              // Send follow-up email
              sendCoachingFollowUpEmail(noteData);
            } else {
              // Reload previous notes
              loadPreviousNotes(noteData.counselorEmail);
            }
          } else {
            // Show error message
            document.getElementById('errorMessage').style.display = 'block';
            document.getElementById('errorMessage').textContent =
result.message || 'An error occurred while saving the coaching note. Please
try again.';
          }
        })
        .withFailureHandler(function(error) {
          // Reset button state
          saveButton.textContent = originalText;
          saveButton.disabled = false;

          // Show error message
          document.getElementById('errorMessage').style.display = 'block';
          document.getElementById('errorMessage').textContent = error.message
|| 'An error occurred while saving the coaching note. Please try again.';
```

```
    })
    .saveCoachingNote(noteData);
}

// Send follow-up email to counselor
function sendCoachingFollowUpEmail(noteData) {
  google.script.run
    .withSuccessHandler(function(result) {
      if (result.success) {
        alert('Follow-up email sent successfully.');
      } else {
        alert('Error sending follow-up email: ' + result.message);
      }

      // Reload previous notes
      loadPreviousNotes(noteData.counselorEmail);
    })
    .withFailureHandler(function(error) {
      alert('Error sending follow-up email: ' + error.message);

      // Reload previous notes
      loadPreviousNotes(noteData.counselorEmail);
    })
    .sendCoachingFollowUpEmail(noteData);
}

// Handle errors
function handleError(error) {
  console.error('Error:', error);

  // Show error message
  document.getElementById('errorMessage').style.display = 'block';
  document.getElementById('successMessage').style.display = 'none';
```

```
    document.getElementById('errorMessage').textContent = error.message ||
'An error occurred. Please try again.';
   }
 </script>
</body>
</html>
<!DOCTYPE html>
<html>
<head>
 <base target="_top">
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <link
href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500;700&fam
ily=Poppins:wght@500;600;700&display=swap" rel="stylesheet">
 <style>
   /* Variables for 988 Lifeline LGBTQIA+ Services color scheme */
   :root {
     /* Primary Colors */
     --trevor-orange: #FF786E;
     --deep-blue: #001A4E;
     --purple: #9A3499;
     --teal: #137F6A;

     /* Secondary Colors */
     --light-blue: #4F52DE;
     --soft-yellow: #FFAD8D;
     --lavender: #B3AE4A;
     --light-purple: #F54AC;

     /* Tertiary Colors */
     --soft-green: #BAE2CE;
     --pale-yellow: #FFF2DF;
```

```
    --light-pink: #FBCBBE;
    --soft-lavender: #D1CFCC;

    /* Gradients */
    --primary-gradient: linear-gradient(135deg, var(--trevor-orange),
var(--light-pink));
    --calm-gradient: linear-gradient(135deg, var(--light-blue),
var(--soft-green));
    --support-gradient: linear-gradient(45deg, var(--purple),
var(--light-purple));
    --background-gradient: linear-gradient(180deg, var(--pale-yellow),
#FFFFFF);

    /* Spacing */
    --spacing-xs: 4px;
    --spacing-sm: 8px;
    --spacing-md: 16px;
    --spacing-lg: 24px;
    --spacing-xl: 32px;

    /* Borders */
    --border-radius-sm: 8px;
    --border-radius-md: 12px;
    --border-radius-lg: 16px;
  }

  body {
    font-family: 'Roboto', sans-serif;
    margin: 0;
    padding: 0;
    background-color: var(--pale-yellow);
    color: var(--deep-blue);
  }
```

```css
.container {
  max-width: 800px;
  margin: 0 auto;
  padding: var(--spacing-md);
}

.header {
  margin-bottom: var(--spacing-lg);
  text-align: center;
}

h1, h2, h3, h4 {
  font-family: 'Poppins', sans-serif;
  color: var(--deep-blue);
}

.content {
  background: white;
  border-radius: var(--border-radius-md);
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.05);
  padding: var(--spacing-lg);
  margin-bottom: var(--spacing-lg);
}

.protocol-section {
  margin-bottom: var(--spacing-xl);
}

.protocol-title {
  font-family: 'Poppins', sans-serif;
  font-size: 1.3rem;
  margin-bottom: var(--spacing-md);
```

```
  color: var(--purple);
  border-bottom: 2px solid var(--pale-yellow);
  padding-bottom: var(--spacing-xs);
}

.protocol-description {
  margin-bottom: var(--spacing-md);
  line-height: 1.5;
}

.steps-list {
  background-color: #f9f9f9;
  padding: var(--spacing-md);
  border-radius: var(--border-radius-sm);
  margin-bottom: var(--spacing-md);
}

.step-item {
  margin-bottom: var(--spacing-md);
  padding-left: var(--spacing-md);
  border-left: 3px solid var(--trevor-orange);
}

.step-title {
  font-weight: 500;
  margin-bottom: var(--spacing-xs);
}

.step-description {
  font-size: 0.95rem;
  color: #333;
}
```

```css
.alert-box {
  background-color: var(--light-pink);
  border-left: 4px solid var(--trevor-orange);
  padding: var(--spacing-md);
  margin-bottom: var(--spacing-md);
  border-radius: 0 var(--border-radius-sm) var(--border-radius-sm) 0;
}

.alert-title {
  font-weight: 500;
  margin-bottom: var(--spacing-xs);
  color: var(--deep-blue);
}

.alert-text {
  color: var(--deep-blue);
}

.tip-box {
  background-color: var(--soft-green);
  border-left: 4px solid var(--teal);
  padding: var(--spacing-md);
  margin-bottom: var(--spacing-md);
  border-radius: 0 var(--border-radius-sm) var(--border-radius-sm) 0;
}

.tip-title {
  font-weight: 500;
  margin-bottom: var(--spacing-xs);
  color: var(--deep-blue);
}

.tip-text {
```

```
  color: var(--deep-blue);
}

.resources-list {
  margin-top: var(--spacing-md);
}

.resource-item {
  margin-bottom: var(--spacing-md);
  display: flex;
  align-items: flex-start;
}

.resource-icon {
  width: 24px;
  height: 24px;
  background-color: var(--soft-lavender);
  border-radius: 50%;
  display: flex;
  align-items: center;
  justify-content: center;
  margin-right: var(--spacing-sm);
  flex-shrink: 0;
}

.resource-content {
  flex: 1;
}

.resource-title {
  font-weight: 500;
  margin-bottom: var(--spacing-xs);
}
```

```css
.resource-description {
  font-size: 0.9rem;
  color: #333;
}

.table-of-contents {
  background-color: #f9f9f9;
  padding: var(--spacing-md);
  border-radius: var(--border-radius-sm);
  margin-bottom: var(--spacing-lg);
}

.toc-title {
  font-weight: 500;
  margin-bottom: var(--spacing-sm);
}

.toc-list {
  list-style-type: none;
  padding-left: var(--spacing-md);
}

.toc-item {
  margin-bottom: var(--spacing-xs);
}

.toc-link {
  color: var(--light-blue);
  text-decoration: none;
}

.toc-link:hover {
```

```
    text-decoration: underline;
  }

  .back-to-top {
    display: block;
    text-align: right;
    margin-top: var(--spacing-md);
    font-size: 0.9rem;
    color: var(--light-blue);
    text-decoration: none;
  }

  .back-to-top:hover {
    text-decoration: underline;
  }
 </style>
</head>
<body>
 <div class="container">
  <div class="header">
    <h1>Crisis Protocols</h1>
    <p>Standard procedures for managing crisis situations in the 988 Lifeline
LGBTQIA+ Crisis Support System</p>
  </div>

  <div class="table-of-contents">
    <div class="toc-title">Table of Contents</div>
    <ul class="toc-list">
      <li class="toc-item"><a href="#suicide-risk" class="toc-link">Suicide
Risk Assessment</a></li>
      <li class="toc-item"><a href="#active-rescue" class="toc-link">Active
Rescue Procedure</a></li>
```

```
      <li class="toc-item"><a href="#lgbtqia-specific"
class="toc-link">LGBTQIA+ Specific Considerations</a></li>
      <li class="toc-item"><a href="#safety-planning" class="toc-link">Safety
Planning</a></li>
      <li class="toc-item"><a href="#resources" class="toc-link">Crisis
Resources</a></li>
    </ul>
  </div>

  <div class="content">
    <div id="suicide-risk" class="protocol-section">
      <h2 class="protocol-title">Suicide Risk Assessment</h2>

      <div class="protocol-description">
        <p>The suicide risk assessment protocol helps counselors evaluate
callers' immediate risk level and determine appropriate interventions. This
structured approach ensures consistent evaluation while maintaining a
supportive and caring conversation.</p>
      </div>

      <div class="alert-box">
        <div class="alert-title">Important Reminder</div>
        <div class="alert-text">Always prioritize building rapport and
connection before direct risk assessment questions. Shift into assessment
naturally within the conversation, maintaining empathy throughout.</div>
      </div>

      <div class="steps-list">
        <div class="step-item">
          <div class="step-title">1. Assess Suicidal Ideation</div>
          <div class="step-description">
            <p>Begin with general questions about thoughts of suicide or
self-harm. Examples:</p>
```

```
            <ul>
              <li>"I'm concerned about you. Are you having thoughts of
suicide?"</li>
              <li>"Sometimes when people feel this way, they think about
suicide. Is that something you're thinking about?"</li>
            </ul>
          </div>
        </div>

        <div class="step-item">
          <div class="step-title">2. Evaluate Plan and Means</div>
          <div class="step-description">
            <p>If ideation is present, assess for specific plan and access to
means:</p>
            <ul>
              <li>"Do you have a plan for how you would end your life?"</li>
              <li>"Do you have access to [specific means mentioned]?"</li>
            </ul>
          </div>
        </div>

        <div class="step-item">
          <div class="step-title">3. Determine Timeframe and Intent</div>
          <div class="step-description">
            <p>Evaluate immediacy of risk and strength of intent:</p>
            <ul>
              <li>"When are you thinking about doing this?"</li>
              <li>"How strong is your desire to act on these thoughts right
now?"</li>
            </ul>
          </div>
        </div>
```

```
        <div class="step-item">
          <div class="step-title">4. Identify Protective Factors</div>
          <div class="step-description">
            <p>Explore reasons for living and supports:</p>
            <ul>
              <li>"What has kept you going until now?"</li>
              <li>"Who in your life would you feel comfortable reaching out
to?"</li>
            </ul>
          </div>
        </div>

        <div class="step-item">
          <div class="step-title">5. Determine Risk Level</div>
          <div class="step-description">
            <p>Based on assessment, categorize risk as:</p>
            <ul>
              <li><strong>Low Risk:</strong> Thoughts but no plan, strong
protective factors</li>
              <li><strong>Moderate Risk:</strong> Thoughts with vague plan,
some protective factors</li>
              <li><strong>High Risk:</strong> Specific plan, access to means,
imminent timeframe, few protective factors</li>
            </ul>
          </div>
        </div>
      </div>

      <div class="tip-box">
        <div class="tip-title">LGBTQIA+ Sensitivity Tip</div>
        <div class="tip-text">When assessing risk with LGBTQIA+ callers, be
mindful that experiences of rejection, discrimination, or issues related to
```

```
gender and sexual identity may be contributing factors. Create space for these specific concerns while avoiding assumptions.</div>
      </div>

      <a href="#" class="back-to-top">Back to Top</a>
    </div>

    <div id="active-rescue" class="protocol-section">
      <h2 class="protocol-title">Active Rescue Procedure</h2>

      <div class="protocol-description">
        <p>The active rescue protocol outlines steps to take when a caller is at imminent risk of suicide and requires emergency intervention. This procedure balances the need for safety with respect for the caller's autonomy when possible.</p>
      </div>

      <div class="alert-box">
        <div class="alert-title">Critical Information</div>
        <div class="alert-text">Active rescue should be initiated when a caller is at imminent risk of suicide with intent, plan, and means, and is unwilling or unable to take steps to ensure their immediate safety.</div>
      </div>

      <div class="steps-list">
        <div class="step-item">
          <div class="step-title">1. Confirm Immediate Risk</div>
          <div class="step-description">
            <p>Verify that the caller presents with:</p>
            <ul>
              <li>Current suicidal ideation</li>
              <li>Specific and lethal plan</li>
              <li>Access to means</li>
```

```
              <li>Intent to act imminently</li>
              <li>Inability to develop a safe alternative</li>
            </ul>
          </div>
        </div>

        <div class="step-item">
          <div class="step-title">2. Attempt Collaborative Safety
Planning</div>
          <div class="step-description">
            <p>Before moving to non-consensual rescue:</p>
            <ul>
              <li>Try to develop a safety plan collaboratively</li>
              <li>Explore removal of means or having someone stay with
them</li>
              <li>Discuss voluntary emergency services engagement</li>
            </ul>
          </div>
        </div>

        <div class="step-item">
          <div class="step-title">3. Gather Location Information</div>
          <div class="step-description">
            <p>If a rescue is necessary, obtain:</p>
            <ul>
              <li>Caller's current location (address, city, state)</li>
              <li>Caller's name</li>
              <li>Description of residence if applicable</li>
              <li>Phone number they're calling from</li>
            </ul>
            <p>Be transparent about why you need this information.</p>
          </div>
        </div>
```

```
        <div class="step-item">
          <div class="step-title">4. Involve Supervisor</div>
          <div class="step-description">
            <p>Alert your supervisor while maintaining the call connection.
They will:</p>
            <ul>
              <li>Confirm risk assessment and rescue necessity</li>
              <li>Contact emergency services</li>
              <li>Document the intervention</li>
            </ul>
          </div>
        </div>

        <div class="step-item">
          <div class="step-title">5. Stay Connected</div>
          <div class="step-description">
            <p>Maintain call connection until:</p>
            <ul>
              <li>Emergency services arrive</li>
              <li>Another support person assumes responsibility</li>
              <li>Risk level significantly decreases</li>
            </ul>
          </div>
        </div>

        <div class="step-item">
          <div class="step-title">6. Document Thoroughly</div>
          <div class="step-description">
            <p>Complete detailed documentation including:</p>
            <ul>
              <li>Risk assessment details</li>
              <li>Attempts at collaborative safety planning</li>
```

```
                <li>Time emergency services were contacted</li>
                <li>Information provided to emergency services</li>
                <li>Outcome of the intervention</li>
              </ul>
            </div>
          </div>
        </div>

        <div class="tip-box">
          <div class="tip-title">LGBTQIA+ Safety Consideration</div>
          <div class="tip-text">Be aware that emergency services interactions can sometimes pose additional risks for LGBTQIA+ individuals, particularly transgender and gender-diverse people. When possible, provide responders with information about appropriate name/pronouns and any specific safety concerns.</div>
        </div>

        <a href="#" class="back-to-top">Back to Top</a>
      </div>

      <div id="lgbtqia-specific" class="protocol-section">
        <h2 class="protocol-title">LGBTQIA+ Specific Considerations</h2>

        <div class="protocol-description">
          <p>This protocol outlines important considerations when supporting LGBTQIA+ individuals in crisis. Understanding these unique factors helps provide culturally responsive care that acknowledges specific stressors and experiences.</p>
        </div>

        <div class="tip-box">
          <div class="tip-title">Affirmation and Validation</div>
```

```
        <div class="tip-text">Always use callers' stated names and pronouns. Affirm their identity as valid and worthy of respect, regardless of others' reactions or treatment. This foundation of respect is essential for effective crisis support.</div>
      </div>

      <div class="steps-list">
        <div class="step-item">
          <div class="step-title">1. Identity-Based Stressors</div>
          <div class="step-description">
            <p>Be aware of specific stressors that may contribute to crisis:</p>
            <ul>
              <li>Family rejection or non-acceptance</li>
              <li>Housing insecurity or homelessness</li>
              <li>Discrimination in employment, healthcare, or education</li>
              <li>Minority stress and internalized negative messages</li>
              <li>Gender dysphoria and transition-related challenges</li>
              <li>Experiences of bullying, harassment, or violence</li>
            </ul>
          </div>
        </div>

        <div class="step-item">
          <div class="step-title">2. Safety Assessment Considerations</div>
          <div class="step-description">
            <p>When assessing safety, include LGBTQIA+-specific factors:</p>
            <ul>
              <li>Safety of home environment regarding identity acceptance</li>
              <li>Access to affirming healthcare providers</li>
              <li>Community resources specifically for LGBTQIA+ individuals</li>
```

```
              <li>Protective factors within LGBTQIA+ community
connections</li>
              <li>Safety concerns with authorities or emergency services</li>
            </ul>
          </div>
        </div>

        <div class="step-item">
          <div class="step-title">3. Resource Navigation</div>
          <div class="step-description">
            <p>Be prepared to connect callers with specialized resources:</p>
            <ul>
              <li>LGBTQIA+-affirming mental health providers</li>
              <li>Transgender healthcare resources</li>
              <li>LGBTQIA+ youth shelters and housing programs</li>
              <li>Legal services for discrimination issues</li>
              <li>Online and in-person LGBTQIA+ support communities</li>
            </ul>
          </div>
        </div>

        <div class="step-item">
          <div class="step-title">4. Intersectionality Awareness</div>
          <div class="step-description">
            <p>Recognize how multiple identities interact and impact
crisis:</p>
            <ul>
              <li>Consider racial, cultural, religious, and disability
intersections</li>
              <li>Acknowledge varying access to resources based on
socioeconomic status</li>
              <li>Understand how immigration status may affect service
access</li>
```

```
              <li>Respect cultural and religious contexts while supporting
identity</li>
            </ul>
          </div>
        </div>
      </div>

      <div class="alert-box">
        <div class="alert-title">Language and Terminology</div>
        <div class="alert-text">Follow the caller's lead in language about
their identity. If unsure, ask respectfully how they describe their identity
rather than making assumptions. Staying current with terminology is important,
but individual preferences always take precedence.</div>
      </div>

      <a href="#" class="back-to-top">Back to Top</a>
    </div>

    <div id="safety-planning" class="protocol-section">
      <h2 class="protocol-title">Safety Planning</h2>

      <div class="protocol-description">
        <p>Safety planning is a collaborative process to develop concrete
strategies for managing suicidal thoughts and crises. This protocol outlines
key components of effective safety planning that can be tailored to each
caller's unique circumstances.</p>
      </div>

      <div class="steps-list">
        <div class="step-item">
          <div class="step-title">1. Recognize Warning Signs</div>
          <div class="step-description">
```

```
            <p>Help the caller identify personal warning signs of an
impending crisis:</p>
            <ul>
              <li>Thoughts: "What thoughts come up when you start feeling
suicidal?"</li>
              <li>Emotions: "What emotions signal you might be heading toward
crisis?"</li>
              <li>Behaviors: "What do you notice yourself doing
differently?"</li>
              <li>Physical sensations: "What body sensations tell you you're
starting to struggle?"</li>
              <li>Social changes: "How do your interactions with others
change?"</li>
            </ul>
          </div>
        </div>

        <div class="step-item">
          <div class="step-title">2. Internal Coping Strategies</div>
          <div class="step-description">
            <p>Develop self-managed coping techniques:</p>
            <ul>
              <li>Grounding exercises (5-4-3-2-1 senses technique, deep
breathing)</li>
              <li>Physical activities (walking, stretching, exercise)</li>
              <li>Distraction methods (music, creative activities,
games)</li>
              <li>Emotional regulation strategies (journaling,
meditation)</li>
              <li>Self-compassion practices</li>
            </ul>
          </div>
        </div>
```

```
        <div class="step-item">
          <div class="step-title">3. Social Connections and Settings</div>
          <div class="step-description">
            <p>Identify people and places that provide distraction and
comfort:</p>
            <ul>
              <li>Safe social environments (coffee shops, libraries,
community centers)</li>
              <li>People who can provide company without discussing the
crisis</li>
              <li>Structured social activities or groups</li>
              <li>Online communities when in-person options aren't
available</li>
            </ul>
          </div>
        </div>

        <div class="step-item">
          <div class="step-title">4. Contact Support Persons</div>
          <div class="step-description">
            <p>Identify specific people who can help during a crisis:</p>
            <ul>
              <li>Create a prioritized list with contact information</li>
              <li>Include at least 3-5 contacts if possible</li>
              <li>Discuss what the caller feels comfortable sharing with each
person</li>
              <li>Consider different types of support (emotional support,
practical help)</li>
            </ul>
          </div>
        </div>
```

```
        <div class="step-item">
          <div class="step-title">5. Professional Resources</div>
          <div class="step-description">
            <p>Compile crisis services and professional support options:</p>
            <ul>
              <li>Crisis hotlines and text lines (988, Trevor Project)</li>
              <li>Local mental health crisis services</li>
              <li>Mental health providers' crisis protocols</li>
              <li>Nearest emergency departments</li>
              <li>Warm lines for non-emergency support</li>
            </ul>
          </div>
        </div>

        <div class="step-item">
          <div class="step-title">6. Making the Environment Safe</div>
          <div class="step-description">
            <p>Develop strategies to reduce access to means:</p>
            <ul>
              <li>Identify specific lethal means that should be removed or
secured</li>
              <li>Create concrete steps for means removal (who can help,
where items can be stored)</li>
              <li>Consider temporary storage options for dangerous items</li>
              <li>Address prescription medication safety if relevant</li>
            </ul>
          </div>
        </div>

        <div class="step-item">
          <div class="step-title">7. Reasons for Living</div>
          <div class="step-description">
```

```
            <p>Help the caller identify personal motivations to stay
alive:</p>
            <ul>
              <li>Relationships (people, pets)</li>
              <li>Future goals and aspirations</li>
              <li>Values and beliefs</li>
              <li>Previous successes in overcoming difficult times</li>
              <li>Small pleasures and moments of joy</li>
            </ul>
          </div>
        </div>
      </div>

      <div class="tip-box">
        <div class="tip-title">Documentation Tip</div>
        <div class="tip-text">Encourage the caller to document their safety
plan in a format that works for them - written, digital, or as a phone note.
Having the plan accessible during crisis moments is crucial for its
effectiveness.</div>
      </div>

      <a href="#" class="back-to-top">Back to Top</a>
    </div>

    <div id="resources" class="protocol-section">
      <h2 class="protocol-title">Crisis Resources</h2>

      <div class="protocol-description">
        <p>This section provides an overview of key resources to share with
callers in crisis. These resources are organized by type and include
LGBTQIA+-specific options when available.</p>
      </div>
```

```
      <div class="resources-list">
        <div class="resource-item">
          <div class="resource-icon">📞</div>
          <div class="resource-content">
            <div class="resource-title">National Crisis Hotlines</div>
            <div class="resource-description">
              <ul>
                <li><strong>988 Suicide & Crisis Lifeline:</strong> Call or text 988 (Available 24/7)</li>
                <li><strong>Trevor Project:</strong> 1-866-488-7386 (LGBTQ youth crisis support, 24/7)</li>
                <li><strong>Trans Lifeline:</strong> 1-877-565-8860 (Peer support for trans people)</li>
                <li><strong>Crisis Text Line:</strong> Text HOME to 741741 (24/7 text support)</li>
              </ul>
            </div>
          </div>
        </div>

        <div class="resource-item">
          <div class="resource-icon">🏠</div>
          <div class="resource-content">
            <div class="resource-title">Housing Resources</div>
            <div class="resource-description">
              <ul>
                <li><strong>True Colors United:</strong> Resources for LGBTQ youth experiencing homelessness</li>
                <li><strong>National Runaway Safeline:</strong> 1-800-RUNAWAY (1-800-786-2929)</li>
                <li><strong>LGBTQ-Affirming Shelters Directory:</strong> Available through findhelp.org</li>
              </ul>
```

```
            </div>
          </div>
        </div>

        <div class="resource-item">
          <div class="resource-icon">⚕ </div>
          <div class="resource-content">
            <div class="resource-title">Healthcare Resources</div>
            <div class="resource-description">
              <ul>
                <li><strong>GLMA Provider Directory:</strong> LGBTQ-affirming healthcare providers</li>
                <li><strong>National Alliance on Mental Illness (NAMI):</strong> 1-800-950-NAMI (6264)</li>
                <li><strong>RAD Remedy:</strong> Connects trans, gender non-conforming, and intersex people to compassionate healthcare providers</li>
              </ul>
            </div>
          </div>
        </div>

        <div class="resource-item">
          <div class="resource-icon">⚖️ </div>
          <div class="resource-content">
            <div class="resource-title">Legal Resources</div>
            <div class="resource-description">
              <ul>
                <li><strong>Lambda Legal:</strong> Legal support for LGBTQ people and those living with HIV</li>
                <li><strong>Transgender Law Center:</strong> Legal information and resources for transgender people</li>
                <li><strong>National Center for Transgender Equality:</strong> ID document change information</li>
```

```
              </ul>
            </div>
          </div>
        </div>

        <div class="resource-item">
          <div class="resource-icon">🌐</div>
          <div class="resource-content">
            <div class="resource-title">Online Communities</div>
            <div class="resource-description">
              <ul>
                <li><strong>Trevor Space:</strong> Online community for LGBTQ youth ages 13-24</li>
                <li><strong>Q Chat Space:</strong> Live, chat-based support groups for LGBTQ teens</li>
                <li><strong>Reddit Communities:</strong> r/LGBTeens, r/asktransgender, r/nonbinary (age-appropriate options)</li>
              </ul>
            </div>
          </div>
        </div>
      </div>

      <div class="alert-box">
        <div class="alert-title">Resource Updates</div>
        <div class="alert-text">Resources may change over time. Always verify contact information and availability before sharing with callers. Report any outdated information to your supervisor for system-wide updates.</div>
      </div>

      <a href="#" class="back-to-top">Back to Top</a>
    </div>
  </div>
```

```
</div>
 <script>
  // Handle "Back to Top" links
  document.querySelectorAll('.back-to-top').forEach(function(link) {
    link.addEventListener('click', function(e) {
      e.preventDefault();
      window.scrollTo({
        top: 0,
        behavior: 'smooth'
      });
    });
  });

  // Handle table of contents links
  document.querySelectorAll('.toc-link').forEach(function(link) {
    link.addEventListener('click', function(e) {
      e.preventDefault();
      const targetId = this.getAttribute('href').substring(1);
      const targetElement = document.getElementById(targetId);

      if (targetElement) {
        window.scrollTo({
          top: targetElement.offsetTop - 20,
          behavior: 'smooth'
        });
      }
    });
  });
 </script>
</body>
</html>
<!DOCTYPE html>
<html>
```

```
<head>
 <base target="_top">
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <link
href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500;700&fam
ily=Poppins:wght@500;600;700&display=swap" rel="stylesheet">
 <style>
   /* Variables for Trevor Project color scheme */
   :root {
     /* Primary Colors */
     --trevor-orange: #FF786E;
     --deep-blue: #001A4E;
     --purple: #9A3499;
     --teal: #137F6A;

     /* Secondary Colors */
     --light-blue: #4F52DE;
     --soft-yellow: #FFAD8D;
     --lavender: #B3AE4A;
     --light-purple: #F54AC;

     /* Tertiary Colors */
     --soft-green: #BAE2CE;
     --pale-yellow: #FFF2DF;
     --light-pink: #FBCBBE;
     --soft-lavender: #D1CFCC;

     /* Gradients */
     --primary-gradient: linear-gradient(135deg, var(--trevor-orange),
var(--light-pink));
     --calm-gradient: linear-gradient(135deg, var(--light-blue),
var(--soft-green));
```

```
    --support-gradient: linear-gradient(45deg, var(--purple),
var(--light-purple));
    --background-gradient: linear-gradient(180deg, var(--pale-yellow),
#FFFFFF);

    /* Spacing */
    --spacing-xs: 4px;
    --spacing-sm: 8px;
    --spacing-md: 16px;
    --spacing-lg: 24px;
    --spacing-xl: 32px;

    /* Borders */
    --border-radius-sm: 8px;
    --border-radius-md: 12px;
    --border-radius-lg: 16px;
  }

  body {
    font-family: 'Roboto', sans-serif;
    margin: 0;
    padding: 0;
    background: var(--background-gradient);
    color: var(--deep-blue);
  }

  h1, h2, h3, h4 {
    font-family: 'Poppins', sans-serif;
    color: var(--deep-blue);
  }

  .container {
    max-width: 1200px;
```

```
  margin: 0 auto;
  padding: var(--spacing-md);
}

.dashboard-header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  margin-bottom: var(--spacing-lg);
}

.welcome-section h1 {
  margin-bottom: var(--spacing-xs);
}

.date {
  color: #666;
}

.dashboard-grid {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(350px, 1fr));
  gap: var(--spacing-md);
  margin-bottom: var(--spacing-lg);
}

.dashboard-card {
  background: white;
  border-radius: var(--border-radius-md);
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.05);
  overflow: hidden;
  position: relative;
}
```

```css
.dashboard-card::before {
  content: '';
  position: absolute;
  left: 0;
  top: 0;
  height: 100%;
  width: 3px;
  background: var(--trevor-orange);
}

.card-header {
  padding: var(--spacing-md);
  border-bottom: 1px solid #eee;
  display: flex;
  justify-content: space-between;
  align-items: center;
}

.card-header h3 {
  margin: 0;
}

.card-actions {
  display: flex;
  gap: var(--spacing-sm);
}

.card-content {
  padding: var(--spacing-md);
}

.metrics-overview {
```

```
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(150px, 1fr));
  gap: var(--spacing-md);
  margin-bottom: var(--spacing-lg);
  padding: var(--spacing-md);
  border-radius: var(--border-radius-md);
  color: white;
}

.metric-card {
  text-align: center;
  padding: var(--spacing-md);
  background: rgba(255, 255, 255, 0.2);
  border-radius: var(--border-radius-sm);
}

.metric-card h3 {
  margin-top: 0;
  margin-bottom: var(--spacing-xs);
  color: white;
  font-size: 16px;
}

.metric-value {
  font-size: 28px;
  font-weight: 700;
  margin-bottom: var(--spacing-xs);
}

.metric-target {
  font-size: 14px;
  opacity: 0.8;
}
```

```css
.alert-list {
  list-style: none;
  padding: 0;
  margin: 0;
}

.alert-item {
  padding: var(--spacing-sm);
  border-bottom: 1px solid #eee;
  display: flex;
  align-items: center;
}

.alert-item:last-child {
  border-bottom: none;
}

.alert-severity {
  width: 12px;
  height: 12px;
  border-radius: 50%;
  margin-right: var(--spacing-sm);
}

.severity-high {
  background-color: var(--light-purple);
}

.severity-medium {
  background-color: var(--soft-yellow);
}
```

```css
.severity-low {
  background-color: var(--light-blue);
}

.task-list {
  list-style: none;
  padding: 0;
  margin: 0;
}

.task-item {
  padding: var(--spacing-sm);
  border-bottom: 1px solid #eee;
  display: flex;
  align-items: center;
}

.task-item:last-child {
  border-bottom: none;
}

.task-checkbox {
  margin-right: var(--spacing-sm);
}

.task-priority {
  width: 12px;
  height: 12px;
  border-radius: 50%;
  margin-right: var(--spacing-sm);
}

.priority-high {
```

```
  background-color: var(--light-purple);
}

.priority-medium {
  background-color: var(--soft-yellow);
}

.priority-low {
  background-color: var(--light-blue);
}

.counselor-list {
  list-style: none;
  padding: 0;
  margin: 0;
}

.counselor-item {
  padding: var(--spacing-sm);
  border-bottom: 1px solid #eee;
  display: flex;
  justify-content: space-between;
  align-items: center;
}

.counselor-item:last-child {
  border-bottom: none;
}

.counselor-status {
  font-size: 12px;
  padding: 2px 8px;
  border-radius: 12px;
```

```css
}

.status-active {
  background-color: var(--soft-green);
}

.status-leave {
  background-color: var(--light-pink);
}

.btn {
  border: none;
  padding: var(--spacing-sm) var(--spacing-md);
  border-radius: var(--border-radius-sm);
  font-weight: 500;
  cursor: pointer;
  transition: all 0.2s ease;
}

.btn-primary {
  background: var(--primary-gradient);
  color: white;
}

.btn-primary:hover {
  box-shadow: 0 4px 8px rgba(255, 120, 110, 0.3);
  transform: translateY(-2px);
}

.btn-secondary {
  background: white;
  color: var(--deep-blue);
  border: 1px solid #ddd;
```

```css
}

.btn-secondary:hover {
  background: #f5f5f5;
}

.btn-sm {
  font-size: 12px;
  padding: 4px 8px;
}

.chart-container {
  width: 100%;
  height: 200px;
}

.loading-spinner {
  display: inline-block;
  width: 20px;
  height: 20px;
  border: 3px solid rgba(255, 120, 110, 0.3);
  border-radius: 50%;
  border-top-color: var(--trevor-orange);
  animation: spin 1s linear infinite;
  margin-right: var(--spacing-sm);
}

@keyframes spin {
  to { transform: rotate(360deg); }
}

.status-dot {
  display: inline-block;
```

```
  width: 10px;
  height: 10px;
  border-radius: 50%;
  margin-right: var(--spacing-xs);
}

.status-success {
  background-color: var(--teal);
}

.status-warning {
  background-color: var(--soft-yellow);
}

.status-critical {
  background-color: var(--light-purple);
}

.badge {
  display: inline-block;
  padding: 2px 8px;
  border-radius: 12px;
  font-size: 12px;
  color: white;
  margin-right: var(--spacing-xs);
}

.badge-critical {
  background-color: var(--light-purple);
}

.badge-warning {
  background-color: var(--soft-yellow);
```

```
  }

  .validation-summary {
    display: flex;
    flex-direction: column;
    gap: var(--spacing-sm);
  }

  .validation-metric {
    display: flex;
    justify-content: space-between;
    align-items: center;
  }

  .validation-issues {
    display: flex;
    gap: var(--spacing-sm);
    margin-top: var(--spacing-xs);
  }

  .metric-title {
    font-weight: 500;
  }

  .metric-value {
    font-weight: 700;
  }
 </style>
</head>
<body>
 <div class="container">
   <div class="dashboard-header">
     <div class="welcome-section">
```

```
      <h1>Welcome, <span id="userName">Team Lead</span></h1>
      <p class="date" id="currentDate">Loading date...</p>
    </div>
  </div>

  <div class="metrics-overview" style="background: var(--calm-gradient)">
    <div class="metric-card">
      <h3>Answer Rate</h3>
      <div class="metric-value" id="answerRate">--</div>
      <div class="metric-target">Target: 95%</div>
    </div>
    <div class="metric-card">
      <h3>Quality Score</h3>
      <div class="metric-value" id="qualityScore">--</div>
      <div class="metric-target">Target: 70+</div>
    </div>
    <div class="metric-card">
      <h3>Avg. Conversations</h3>
      <div class="metric-value" id="avgConversations">--</div>
      <div class="metric-target">Target: 8-10</div>
    </div>
    <div class="metric-card">
      <h3>Unplanned Time Off</h3>
      <div class="metric-value" id="utoRate">--</div>
      <div class="metric-target">Target: <5%</div>
    </div>
  </div>

  <div class="dashboard-grid">
    <!-- Active Alerts Card -->
    <div class="dashboard-card">
      <div class="card-header">
        <h3>Active Alerts</h3>
```

```
        <div class="card-actions">
          <button class="btn-primary btn-sm" onclick="createAlert()">New
Alert</button>
        </div>
      </div>
      <div class="card-content">
        <ul class="alert-list" id="alertList">
          <li class="alert-item">
            <span class="alert-severity severity-high"></span>
            <span>Loading alerts...</span>
          </li>
        </ul>
      </div>
    </div>

    <!-- Pending Tasks Card -->
    <div class="dashboard-card">
      <div class="card-header">
        <h3>Pending Tasks</h3>
        <div class="card-actions">
          <button class="btn-primary btn-sm" onclick="createTask()">New
Task</button>
        </div>
      </div>
      <div class="card-content">
        <ul class="task-list" id="taskList">
          <li class="task-item">
            <span class="task-priority priority-medium"></span>
            <span>Loading tasks...</span>
          </li>
        </ul>
      </div>
    </div>
```

```
    <!-- Performance Trends Card -->
    <div class="dashboard-card">
      <div class="card-header">
        <h3>Performance Trends</h3>
        <div class="card-actions">
          <button class="btn-secondary btn-sm"
onclick="changeTimeframe('week')">Week</button>
          <button class="btn-secondary btn-sm"
onclick="changeTimeframe('month')">Month</button>
        </div>
      </div>
      <div class="card-content">
        <div id="performanceChart" class="chart-container">
          <p>Loading chart data...</p>
        </div>
      </div>
    </div>

    <!-- Active Counselors Card -->
    <div class="dashboard-card">
      <div class="card-header">
        <h3>Team Status</h3>
        <div class="card-actions">
          <button class="btn-secondary btn-sm"
onclick="refreshTeamStatus()">Refresh</button>
        </div>
      </div>
      <div class="card-content">
        <ul class="counselor-list" id="counselorList">
          <li class="counselor-item">
            <span>Loading team data...</span>
          </li>
```

```
        </ul>
      </div>
    </div>

    <!-- Quick Actions Card -->
    <div class="dashboard-card">
      <div class="card-header">
        <h3>Quick Actions</h3>
      </div>
      <div class="card-content">
        <div style="display: grid; grid-template-columns: 1fr 1fr; gap:
10px;">
          <button class="btn btn-primary" onclick="initializeShift()">Start
Shift</button>
          <button class="btn btn-primary" onclick="showTimeTracker()">Time
Tracker</button>
          <button class="btn btn-secondary"
onclick="enterDailyMetrics()">Enter Metrics</button>
          <button class="btn btn-secondary"
onclick="scheduleOneOnOne()">Schedule 1:1</button>
        </div>
      </div>
    </div>

    <!-- System Health Card -->
    <div class="dashboard-card">
      <div class="card-header">
        <h3>System Health</h3>
        <div class="card-actions">
          <button class="btn-primary btn-sm"
onclick="runValidation()">Validate Now</button>
        </div>
      </div>
```

```
      <div class="card-content">
        <div class="validation-summary">
          <div class="validation-metric">
            <span class="metric-title">System Health</span>
            <span class="metric-value" id="validationStatus">--</span>
          </div>
          <div class="validation-metric">
            <span class="metric-title">Last Validation</span>
            <span class="metric-value" id="lastValidation">--</span>
          </div>
          <div class="validation-issues">
            <div class="issue-badge" id="criticalIssues"
style="display:none;">
              <span class="badge badge-critical">0 Critical</span>
            </div>
            <div class="issue-badge" id="warningIssues"
style="display:none;">
              <span class="badge badge-warning">0 Warnings</span>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
  <script>
    // Initialize when the page loads
    document.addEventListener('DOMContentLoaded', function() {
      loadCurrentUser();
      setCurrentDate();
      loadDashboardData();
      loadValidationStatus();
    });
```

```
    // Set the current date
    function setCurrentDate() {
      const now = new Date();
      const options = { weekday: 'long', year: 'numeric', month: 'long', day:
'numeric' };
      document.getElementById('currentDate').innerText =
now.toLocaleDateString('en-US', options);
    }

    // Load current user information
    function loadCurrentUser() {
      google.script.run
        .withSuccessHandler(updateUserDisplay)
        .withFailureHandler(handleError)
        .getCurrentUserInfo();
    }

    // Update the user display
    function updateUserDisplay(userInfo) {
      document.getElementById('userName').innerText = userInfo.name || 'Team
Lead';
    }

    // Load dashboard data
    function loadDashboardData() {
      // Load metrics
      google.script.run
        .withSuccessHandler(updateMetrics)
        .withFailureHandler(handleError)
        .getDashboardMetrics();

      // Load alerts
```

```
    google.script.run
      .withSuccessHandler(updateAlerts)
      .withFailureHandler(handleError)
      .getActiveAlerts();

    // Load tasks
    google.script.run
      .withSuccessHandler(updateTasks)
      .withFailureHandler(handleError)
      .getPendingTasks();

    // Load performance chart data
    google.script.run
      .withSuccessHandler(updatePerformanceChart)
      .withFailureHandler(handleError)
      .getPerformanceData('week');

    // Load counselor status
    google.script.run
      .withSuccessHandler(updateCounselorList)
      .withFailureHandler(handleError)
      .getTeamStatus();
  }

  // Update metrics display
  function updateMetrics(metrics) {
    if (metrics) {
      document.getElementById('answerRate').innerText = metrics.answerRate ||
'--';
      document.getElementById('qualityScore').innerText =
metrics.qualityScore || '--';
      document.getElementById('avgConversations').innerText =
metrics.avgConversations || '--';
```

```
      document.getElementById('utoRate').innerText = metrics.utoRate || '--';
    }
  }

  // Update alerts list
  function updateAlerts(alerts) {
    const alertList = document.getElementById('alertList');
    alertList.innerHTML = '';

    if (alerts && alerts.length > 0) {
      alerts.forEach(alert => {
        const severityClass =
          alert.severity === 'High' ? 'severity-high' :
          alert.severity === 'Medium' ? 'severity-medium' : 'severity-low';

        const li = document.createElement('li');
        li.className = 'alert-item';
        li.innerHTML = `
          <span class="alert-severity ${severityClass}"></span>
          <span>${alert.message}</span>
        `;
        alertList.appendChild(li);
      });
    } else {
      alertList.innerHTML = '<li class="alert-item"><span>No active
alerts</span></li>';
    }
  }

  // Update tasks list
  function updateTasks(tasks) {
    const taskList = document.getElementById('taskList');
    taskList.innerHTML = '';
```

```
    if (tasks && tasks.length > 0) {
      tasks.forEach(task => {
        const priorityClass =
          task.priority === 'High' ? 'priority-high' :
          task.priority === 'Medium' ? 'priority-medium' : 'priority-low';

        const li = document.createElement('li');
        li.className = 'task-item';
        li.innerHTML = `
          <span class="task-priority ${priorityClass}"></span>
          <span>${task.task}</span>
        `;
        taskList.appendChild(li);
      });
    } else {
      taskList.innerHTML = '<li class="task-item"><span>No pending tasks</span></li>';
    }
  }

  // Update performance chart
  function updatePerformanceChart(data) {
    const chartContainer = document.getElementById('performanceChart');

    if (data) {
      // In a real implementation, this would use a charting library
      chartContainer.innerHTML = '<p>Chart data loaded (visualization would be implemented with a charting library)</p>';
    } else {
      chartContainer.innerHTML = '<p>No performance data available</p>';
    }
  }
```

```
  // Update counselor list
  function updateCounselorList(counselors) {
    const counselorList = document.getElementById('counselorList');
    counselorList.innerHTML = '';

    if (counselors && counselors.length > 0) {
      counselors.forEach(counselor => {
        const statusClass = counselor.status === 'Active' ? 'status-active' :
'status-leave';

        const li = document.createElement('li');
        li.className = 'counselor-item';
        li.innerHTML = `
          <span>${counselor.name}</span>
          <span class="counselor-status
${statusClass}">${counselor.status}</span>
        `;
        counselorList.appendChild(li);
      });
    } else {
      counselorList.innerHTML = '<li class="counselor-item"><span>No
counselor data available</span></li>';
    }
  }

  // Change the timeframe for the performance chart
  function changeTimeframe(timeframe) {
    document.getElementById('performanceChart').innerHTML = '<p>Loading chart
data...</p>';

    google.script.run
      .withSuccessHandler(updatePerformanceChart)
```

```
      .withFailureHandler(handleError)
      .getPerformanceData(timeframe);
  }

  // Refresh team status
  function refreshTeamStatus() {
    document.getElementById('counselorList').innerHTML = '<li
class="counselor-item"><span>Loading team data...</span></li>';

    google.script.run
      .withSuccessHandler(updateCounselorList)
      .withFailureHandler(handleError)
      .getTeamStatus();
  }

  // Create a new alert
  function createAlert() {
    google.script.run
      .withSuccessHandler(function() {
        google.script.run
          .withSuccessHandler(updateAlerts)
          .withFailureHandler(handleError)
          .getActiveAlerts();
      })
      .withFailureHandler(handleError)
      .showAlertForm();
  }

  // Create a new task
  function createTask() {
    google.script.run
      .withSuccessHandler(function() {
        google.script.run
```

```
          .withSuccessHandler(updateTasks)
          .withFailureHandler(handleError)
          .getPendingTasks();
      })
      .withFailureHandler(handleError)
      .showTaskForm();
}

// Initialize a shift
function initializeShift() {
  google.script.run
    .withFailureHandler(handleError)
    .showShiftInitialization();
}

// Show time tracker
function showTimeTracker() {
  google.script.run
    .withFailureHandler(handleError)
    .showTimeTracker();
}

// Enter daily metrics
function enterDailyMetrics() {
  google.script.run
    .withFailureHandler(handleError)
    .showMetricsForm();
}

// Schedule a one-on-one
function scheduleOneOnOne() {
  google.script.run
    .withFailureHandler(handleError)
```

```
        .showOneOnOneNotes();
    }

    // Load validation status
    function loadValidationStatus() {
      google.script.run
        .withSuccessHandler(updateValidationStatus)
        .withFailureHandler(handleError)
        .getValidationStatus();
    }

    // Run validation
    function runValidation() {
      // Show loading state
      document.getElementById('validationStatus').innerHTML = '<span
class="loading-spinner"></span> Validating...';

      google.script.run
        .withSuccessHandler(updateValidationStatus)
        .withFailureHandler(handleError)
        .validateSystemStructure();
    }

    // Update the validation status display
    function updateValidationStatus(status) {
      // Update status indicator
      const statusElement = document.getElementById('validationStatus');
      if (status.success) {
        statusElement.innerHTML = '<span class="status-dot
status-success"></span> Healthy';
      } else if (status.criticalIssues > 0) {
        statusElement.innerHTML = '<span class="status-dot
status-critical"></span> Critical Issues';
```

```
    } else {
      statusElement.innerHTML = '<span class="status-dot
status-warning"></span> Warnings';
    }

    // Update last run time
    if (status.lastRun) {
      document.getElementById('lastValidation').innerText = new
Date(status.lastRun).toLocaleString();
    }

    // Update issue badges
    const criticalElement = document.getElementById('criticalIssues');
    if (status.criticalIssues > 0) {
      criticalElement.style.display = 'inline-block';
      criticalElement.querySelector('.badge').innerText =
status.criticalIssues + ' Critical';
    } else {
      criticalElement.style.display = 'none';
    }

    const warningElement = document.getElementById('warningIssues');
    if (status.warningIssues > 0) {
      warningElement.style.display = 'inline-block';
      warningElement.querySelector('.badge').innerText = status.warningIssues
+ ' Warnings';
    } else {
      warningElement.style.display = 'none';
    }
  }

  // Handle errors
  function handleError(error) {
```

```
    console.error('Error:', error);
   }
 </script>
</body>
</html>
<!DOCTYPE html>
<html>
<head>
 <base target="_top">
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <link
href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500;700&fam
ily=Poppins:wght@500;600;700&display=swap" rel="stylesheet">
 <style>
   /* Variables for 988 Lifeline LGBTQIA+ Services color scheme */
   :root {
     /* Primary Colors */
     --trevor-orange: #FF786E;
     --deep-blue: #001A4E;
     --purple: #9A3499;
     --teal: #137F6A;

     /* Secondary Colors */
     --light-blue: #4F52DE;
     --soft-yellow: #FFAD8D;
     --lavender: #B3AE4A;
     --light-purple: #F54AC;

     /* Tertiary Colors */
     --soft-green: #BAE2CE;
     --pale-yellow: #FFF2DF;
     --light-pink: #FBCBBE;
```

```
    --soft-lavender: #D1CFCC;

    /* Gradients */
    --primary-gradient: linear-gradient(135deg, var(--trevor-orange),
var(--light-pink));
    --calm-gradient: linear-gradient(135deg, var(--light-blue),
var(--soft-green));
    --support-gradient: linear-gradient(45deg, var(--purple),
var(--light-purple));
    --background-gradient: linear-gradient(180deg, var(--pale-yellow),
#FFFFFF);

    /* Spacing */
    --spacing-xs: 4px;
    --spacing-sm: 8px;
    --spacing-md: 16px;
    --spacing-lg: 24px;
    --spacing-xl: 32px;

    /* Borders */
    --border-radius-sm: 8px;
    --border-radius-md: 12px;
    --border-radius-lg: 16px;
  }

  body {
    font-family: 'Roboto', sans-serif;
    margin: 0;
    padding: 0;
    background-color: var(--pale-yellow);
    color: var(--deep-blue);
  }
```

```css
.container {
  max-width: 800px;
  margin: 0 auto;
  padding: var(--spacing-md);
}

.header {
  margin-bottom: var(--spacing-lg);
  text-align: center;
}

h1, h2, h3, h4 {
  font-family: 'Poppins', sans-serif;
  color: var(--deep-blue);
}

.tabs {
  display: flex;
  flex-wrap: wrap;
  border-bottom: 1px solid #ddd;
  margin-bottom: var(--spacing-lg);
}

.tab {
  padding: var(--spacing-sm) var(--spacing-lg);
  cursor: pointer;
  border-bottom: 2px solid transparent;
  transition: all 0.2s ease;
}

.tab.active {
  border-bottom-color: var(--trevor-orange);
  font-weight: 500;
```

```css
}

.tab-content {
  display: none;
}

.tab-content.active {
  display: block;
}

.card {
  background: white;
  border-radius: var(--border-radius-md);
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.05);
  padding: var(--spacing-lg);
  margin-bottom: var(--spacing-lg);
}

.search-box {
  margin-bottom: var(--spacing-lg);
  position: relative;
}

.search-input {
  width: 100%;
  padding: var(--spacing-md);
  padding-left: 40px;
  border: 1px solid #ddd;
  border-radius: var(--border-radius-sm);
  font-size: 16px;
}

.search-icon {
```

```
  position: absolute;
  left: 12px;
  top: 50%;
  transform: translateY(-50%);
  color: #888;
}

.help-item {
  margin-bottom: var(--spacing-lg);
  border-bottom: 1px solid #eee;
  padding-bottom: var(--spacing-md);
}

.help-item:last-child {
  border-bottom: none;
}

.help-title {
  font-weight: 500;
  font-size: 1.1rem;
  margin-bottom: var(--spacing-xs);
  color: var(--deep-blue);
}

.help-description {
  margin-bottom: var(--spacing-sm);
  line-height: 1.5;
}

.help-steps {
  padding-left: 20px;
}
```

```css
.help-steps li {
  margin-bottom: var(--spacing-xs);
}

.help-image {
  max-width: 100%;
  margin: var(--spacing-md) 0;
  border: 1px solid #eee;
  border-radius: var(--border-radius-sm);
}

.help-tag {
  display: inline-block;
  padding: 2px 8px;
  background-color: var(--soft-lavender);
  border-radius: 12px;
  font-size: 12px;
  margin-right: var(--spacing-xs);
  margin-bottom: var(--spacing-xs);
}

.help-tags {
  margin-top: var(--spacing-sm);
}

.tip-box {
  background-color: var(--soft-green);
  border-left: 4px solid var(--teal);
  padding: var(--spacing-md);
  margin: var(--spacing-md) 0;
  border-radius: 0 var(--border-radius-sm) var(--border-radius-sm) 0;
}
```

```css
.tip-title {
  font-weight: 500;
  margin-bottom: var(--spacing-xs);
}

.tip-text {
  font-size: 0.95rem;
}

.contact-section {
  margin-top: var(--spacing-lg);
  padding: var(--spacing-md);
  background-color: #f9f9f9;
  border-radius: var(--border-radius-sm);
  text-align: center;
}

.contact-title {
  font-weight: 500;
  margin-bottom: var(--spacing-sm);
}

.contact-info {
  font-size: 1.1rem;
  color: var(--light-blue);
}

.video-section {
  position: relative;
  padding-bottom: 56.25%; /* 16:9 aspect ratio */
  height: 0;
  overflow: hidden;
  margin: var(--spacing-md) 0;
```

```css
}

.video-placeholder {
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background-color: #eee;
  display: flex;
  justify-content: center;
  align-items: center;
  border-radius: var(--border-radius-sm);
}

.category-list {
  display: flex;
  flex-wrap: wrap;
  gap: var(--spacing-sm);
  margin-bottom: var(--spacing-lg);
}

.category-item {
  padding: var(--spacing-sm) var(--spacing-md);
  background-color: white;
  border-radius: var(--border-radius-sm);
  cursor: pointer;
  transition: all 0.2s;
}

.category-item:hover {
  transform: translateY(-2px);
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
```

```
    }

    .category-item.active {
      background-color: var(--light-blue);
      color: white;
    }
  </style>
</head>
<body>
  <div class="container">
    <div class="header">
      <h1>Help Resources</h1>
      <p>Support documentation for the 988 Lifeline LGBTQIA+ Crisis Support System</p>
    </div>

    <div class="search-box">
      <span class="search-icon">🔍</span>
      <input type="text" class="search-input" id="searchInput" placeholder="Search help articles...">
    </div>

    <div class="category-list" id="categoryList">
      <div class="category-item active" data-category="all">All</div>
      <div class="category-item" data-category="getting-started">Getting Started</div>
      <div class="category-item" data-category="call-tracking">Call Tracking</div>
      <div class="category-item" data-category="team-management">Team Management</div>
      <div class="category-item" data-category="scheduling">Scheduling</div>
      <div class="category-item" data-category="troubleshooting">Troubleshooting</div>
```

```
  </div>

  <div class="tabs">
    <div class="tab active" data-tab="help-articles">Help Articles</div>
    <div class="tab" data-tab="faqs">FAQs</div>
    <div class="tab" data-tab="tutorials">Video Tutorials</div>
  </div>

  <!-- Help Articles Tab -->
  <div id="help-articles" class="tab-content active">
    <div class="card">
      <div id="helpItems">
        <div class="help-item" data-category="getting-started">
          <div class="help-title">Getting Started with the System</div>
          <div class="help-description">
            This guide will help you get familiar with the 988 Lifeline
LGBTQIA+ Crisis Support System and its core features.
          </div>
          <div class="help-steps">
            <ol>
              <li>Log into the Google Sheet using your provided
credentials</li>
              <li>Navigate to the 'Crisis Support' menu in the top menu
bar</li>
              <li>Open the Dashboard to see an overview of the system</li>
              <li>Use the Sidebar for quick access to common functions</li>
            </ol>
          </div>
          <div class="help-tags">
            <span class="help-tag">Getting Started</span>
            <span class="help-tag">New User</span>
          </div>
        </div>
```

```
        <div class="help-item" data-category="call-tracking">
          <div class="help-title">Recording Call Metrics</div>
          <div class="help-description">
            Learn how to properly record and track call metrics for reporting and quality assurance.
          </div>
          <div class="help-steps">
            <ol>
              <li>Click "Enter Daily Metrics" from the Call Metrics menu</li>
              <li>Select the appropriate date for the metrics</li>
              <li>Enter the total number of calls handled during your shift</li>
              <li>Record any risk interventions performed</li>
              <li>Document resources provided and referrals made</li>
              <li>Submit the form to save your metrics</li>
            </ol>
          </div>
          <div class="tip-box">
            <div class="tip-title">Important Tip</div>
            <div class="tip-text">Be sure to enter your metrics at the end of each shift. Accurate data helps us improve our services and secure continued funding.</div>
          </div>
          <div class="help-tags">
            <span class="help-tag">Metrics</span>
            <span class="help-tag">Reporting</span>
            <span class="help-tag">Daily Tasks</span>
          </div>
        </div>

        <div class="help-item" data-category="team-management">
          <div class="help-title">Adding Team Members</div>
```

```
            <div class="help-description">
              This guide explains the process for adding new counselors to the
system.
            </div>
            <div class="help-steps">
              <ol>
                <li>From the Team Management menu, select "Add Counselor"</li>
                <li>Fill out all required fields in the form</li>
                <li>Select the appropriate role and permissions level</li>
                <li>Assign a supervisor to the new counselor</li>
                <li>Submit the form to create the new user</li>
              </ol>
            </div>
            <div class="help-tags">
              <span class="help-tag">Team Management</span>
              <span class="help-tag">Onboarding</span>
              <span class="help-tag">Admin</span>
            </div>
          </div>

          <div class="help-item" data-category="scheduling">
            <div class="help-title">Managing the Weekly Schedule</div>
            <div class="help-description">
              Learn how to create and manage weekly shifts for the crisis
support team.
            </div>
            <div class="help-steps">
              <ol>
                <li>Go to the Schedule menu and select "Manage Schedule"</li>
                <li>Navigate to the desired week using the date controls</li>
                <li>Click the "+" button in any shift cell to assign a
counselor</li>
                <li>Select the counselor from the dropdown menu</li>
```

```
              <li>Add any notes relevant to the shift</li>
              <li>Use "Initialize Week" to set up a standard week
template</li>
            </ol>
          </div>
          <div class="help-tags">
            <span class="help-tag">Scheduling</span>
            <span class="help-tag">Team Management</span>
            <span class="help-tag">Admin</span>
          </div>
        </div>

        <div class="help-item" data-category="troubleshooting">
          <div class="help-title">Troubleshooting Common Issues</div>
          <div class="help-description">
            Solutions for common issues that may arise when using the system.
          </div>
          <ul class="help-steps">
            <li><strong>System Loading Slowly:</strong> Try refreshing the
page or clearing your browser cache</li>
            <li><strong>Menu Not Appearing:</strong> Ensure you have the
necessary permissions assigned to your account</li>
            <li><strong>Data Not Saving:</strong> Check your internet
connection and try submitting again</li>
            <li><strong>Forms Not Opening:</strong> Verify that pop-ups are
allowed in your browser settings</li>
            <li><strong>Missing Sheets:</strong> Use the System Validation
tool in Settings to repair missing components</li>
          </ul>
          <div class="help-tags">
            <span class="help-tag">Troubleshooting</span>
            <span class="help-tag">Support</span>
            <span class="help-tag">Common Issues</span>
```

```
          </div>
        </div>
      </div>
    </div>
  </div>

  <!-- FAQs Tab -->
  <div id="faqs" class="tab-content">
    <div class="card">
      <div class="help-item">
        <div class="help-title">How do I reset my password?</div>
        <div class="help-description">
          Password resets are managed through your Google account. Follow
these steps to reset your password:
          <ol class="help-steps">
            <li>Go to accounts.google.com</li>
            <li>Click on "Security" in the left navigation</li>
            <li>Under "Signing in to Google," select "Password"</li>
            <li>Follow the prompts to verify your identity and create a new
password</li>
          </ol>
        </div>
      </div>

      <div class="help-item">
        <div class="help-title">Can I access the system on my mobile
device?</div>
        <div class="help-description">
          Yes, the Crisis Support System is accessible on mobile devices
through the Google Sheets mobile app or a mobile web browser. However, some
advanced features may be easier to use on a desktop computer. We recommend
using a computer for complex tasks like scheduling and reporting.
        </div>
```

```
      </div>

      <div class="help-item">
        <div class="help-title">How do I create a quality review for a counselor?</div>
        <div class="help-description">
          To create a quality review:
          <ol class="help-steps">
            <li>Select "Quality Review" from the Team Management menu</li>
            <li>Choose the counselor you want to review</li>
            <li>Enter the date and interaction ID of the call being reviewed</li>
            <li>Complete all evaluation criteria sections</li>
            <li>Add detailed notes for each section to provide specific feedback</li>
            <li>Submit the review when complete</li>
          </ol>
          The counselor will be able to see their review in their personal dashboard.
        </div>
      </div>

      <div class="help-item">
        <div class="help-title">Who can see the call metrics I enter?</div>
        <div class="help-description">
          Call metrics are visible to team leads, supervisors, and administrators. Individual counselors can see their own metrics but not those of others. Aggregate metrics are used for reporting and quality improvement but are not tied to individual performance evaluations without additional context and review.
        </div>
      </div>
```

```
      <div class="help-item">
        <div class="help-title">How do I export data for external
reporting?</div>
        <div class="help-description">
          To export data:
          <ol class="help-steps">
            <li>Go to the Metrics Report view from the Call Metrics menu</li>
            <li>Set the date range for the data you want to export</li>
            <li>Apply any desired filters (counselor, shift type, etc.)</li>
            <li>Click the "Export" button in the top right corner</li>
            <li>Select your preferred format (CSV or Excel)</li>
            <li>The exported file will download to your computer</li>
          </ol>
        </div>
      </div>
    </div>

    <div class="contact-section">
      <div class="contact-title">Still have questions?</div>
      <div class="contact-info">Contact the system administrator at
support@988lgbtq.org</div>
    </div>
  </div>

  <!-- Video Tutorials Tab -->
  <div id="tutorials" class="tab-content">
    <div class="card">
      <div class="help-item">
        <div class="help-title">Getting Started Tutorial</div>
        <div class="help-description">
          A comprehensive overview of the Crisis Support System and its main
features.
        </div>
```

```
        <div class="video-section">
          <div class="video-placeholder">
            Video Tutorial: Getting Started (10:23)
          </div>
        </div>
      </div>

      <div class="help-item">
        <div class="help-title">Managing Team Schedules</div>
        <div class="help-description">
          Learn how to create and manage weekly shift schedules for your
crisis support team.
        </div>
        <div class="video-section">
          <div class="video-placeholder">
            Video Tutorial: Schedule Management (7:45)
          </div>
        </div>
      </div>

      <div class="help-item">
        <div class="help-title">Quality Review Process</div>
        <div class="help-description">
          A step-by-step walkthrough of conducting effective quality reviews.
        </div>
        <div class="video-section">
          <div class="video-placeholder">
            Video Tutorial: Quality Reviews (12:18)
          </div>
        </div>
      </div>
    </div>
```

```
    <div class="tip-box">
      <div class="tip-title">Video Tip</div>
      <div class="tip-text">Videos are best viewed in full screen mode. Click the full screen icon in the video player controls to expand.</div>
    </div>
  </div>
</div>
 <script>
  // Initialize the page
  function initialize() {
    // Set up tabs
    document.querySelectorAll('.tab').forEach(function(tab) {
      tab.addEventListener('click', function() {
        // Get tab id
        const tabId = tab.getAttribute('data-tab');

        // Update active tab
        document.querySelectorAll('.tab').forEach(tab => tab.classList.remove('active'));
        tab.classList.add('active');

        // Show corresponding content
        document.querySelectorAll('.tab-content').forEach(content => content.classList.remove('active'));
        document.getElementById(tabId).classList.add('active');
      });
    });

    // Set up category filtering
    document.querySelectorAll('.category-item').forEach(function(item) {
      item.addEventListener('click', function() {
        const category = item.getAttribute('data-category');
```

```
            // Update active category
            document.querySelectorAll('.category-item').forEach(item =>
item.classList.remove('active'));
            item.classList.add('active');

            filterHelpItems(category);
        });
    });

    // Set up search functionality
    document.getElementById('searchInput').addEventListener('input',
function() {
        const searchTerm = this.value.toLowerCase();
        const helpItems = document.querySelectorAll('.help-item');

        helpItems.forEach(function(item) {
            const title =
item.querySelector('.help-title').textContent.toLowerCase();
            const description =
item.querySelector('.help-description').textContent.toLowerCase();
            const tags = Array.from(item.querySelectorAll('.help-tag')).map(tag
=> tag.textContent.toLowerCase());

            // Check if item matches search
            const matchesSearch = title.includes(searchTerm) ||
                                  description.includes(searchTerm) ||
                                  tags.some(tag => tag.includes(searchTerm));

            // Only show if matches search and not filtered by category
            const activeCategory =
document.querySelector('.category-item.active').getAttribute('data-category');
            const matchesCategory = activeCategory === 'all' ||
item.getAttribute('data-category') === activeCategory;
```

```
        item.style.display = matchesSearch && matchesCategory ? 'block' :
'none';
      });
    });
  }

  // Filter help items by category
  function filterHelpItems(category) {
    const helpItems = document.querySelectorAll('.help-item');
    const searchTerm =
document.getElementById('searchInput').value.toLowerCase();

    helpItems.forEach(function(item) {
      const itemCategory = item.getAttribute('data-category');

      // Check if item matches search
      const title =
item.querySelector('.help-title').textContent.toLowerCase();
      const description =
item.querySelector('.help-description').textContent.toLowerCase();
      const tags = Array.from(item.querySelectorAll('.help-tag')).map(tag =>
tag.textContent.toLowerCase());
      const matchesSearch = searchTerm === '' ||
                            title.includes(searchTerm) ||
                            description.includes(searchTerm) ||
                            tags.some(tag => tag.includes(searchTerm));

      // Show if matches category and search
      if ((category === 'all' || itemCategory === category) && matchesSearch)
{
        item.style.display = 'block';
      } else {
```

```
        item.style.display = 'none';
      }
    });
  }

  // Initialize when the page loads
  google.script.onload = function() {
    initialize();
  };
 </script>
</body>
</html>
<!DOCTYPE html>
<html>
<head>
 <base target="_top">
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <link
href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500;700&fam
ily=Poppins:wght@500;600;700&display=swap" rel="stylesheet">
 <style>
  /* Variables for 988 Lifeline LGBTQIA+ Services color scheme */
  :root {
    /* Primary Colors */
    --trevor-orange: #FF786E;
    --deep-blue: #001A4E;
    --purple: #9A3499;
    --teal: #137F6A;

    /* Secondary Colors */
    --light-blue: #4F52DE;
    --soft-yellow: #FFAD8D;
```

```
    --lavender: #B3AE4A;
    --light-purple: #F54AC;

    /* Tertiary Colors */
    --soft-green: #BAE2CE;
    --pale-yellow: #FFF2DF;
    --light-pink: #FBCBBE;
    --soft-lavender: #D1CFCC;

    /* Gradients */
    --primary-gradient: linear-gradient(135deg, var(--trevor-orange),
var(--light-pink));
    --calm-gradient: linear-gradient(135deg, var(--light-blue),
var(--soft-green));
    --support-gradient: linear-gradient(45deg, var(--purple),
var(--light-purple));
    --background-gradient: linear-gradient(180deg, var(--pale-yellow),
#FFFFFF);

    /* Spacing */
    --spacing-xs: 4px;
    --spacing-sm: 8px;
    --spacing-md: 16px;
    --spacing-lg: 24px;
    --spacing-xl: 32px;

    /* Borders */
    --border-radius-sm: 8px;
    --border-radius-md: 12px;
    --border-radius-lg: 16px;
  }

  body {
```

```
  font-family: 'Roboto', sans-serif;
  margin: 0;
  padding: 0;
  background-color: var(--pale-yellow);
  color: var(--deep-blue);
}

.container {
  max-width: 800px;
  margin: 0 auto;
  padding: var(--spacing-md);
}

.header {
  margin-bottom: var(--spacing-lg);
  text-align: center;
}

h1, h2, h3, h4 {
  font-family: 'Poppins', sans-serif;
  color: var(--deep-blue);
}

.content {
  background: white;
  border-radius: var(--border-radius-md);
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.05);
  padding: var(--spacing-lg);
  margin-bottom: var(--spacing-lg);
}

.protocol-section {
  margin-bottom: var(--spacing-xl);
```

```
}

.protocol-title {
  font-family: 'Poppins', sans-serif;
  font-size: 1.3rem;
  margin-bottom: var(--spacing-md);
  color: var(--purple);
  border-bottom: 2px solid var(--pale-yellow);
  padding-bottom: var(--spacing-xs);
}

.protocol-description {
  margin-bottom: var(--spacing-md);
  line-height: 1.5;
}

.steps-list {
  background-color: #f9f9f9;
  padding: var(--spacing-md);
  border-radius: var(--border-radius-sm);
  margin-bottom: var(--spacing-md);
}

.step-item {
  margin-bottom: var(--spacing-md);
  padding-left: var(--spacing-md);
  border-left: 3px solid var(--trevor-orange);
}

.step-title {
  font-weight: 500;
  margin-bottom: var(--spacing-xs);
}
```

```css
.step-description {
  font-size: 0.95rem;
  color: #333;
}

.alert-box {
  background-color: var(--light-pink);
  border-left: 4px solid var(--trevor-orange);
  padding: var(--spacing-md);
  margin-bottom: var(--spacing-md);
  border-radius: 0 var(--border-radius-sm) var(--border-radius-sm) 0;
}

.alert-title {
  font-weight: 500;
  margin-bottom: var(--spacing-xs);
  color: var(--deep-blue);
}

.alert-text {
  color: var(--deep-blue);
}

.tip-box {
  background-color: var(--soft-green);
  border-left: 4px solid var(--teal);
  padding: var(--spacing-md);
  margin-bottom: var(--spacing-md);
  border-radius: 0 var(--border-radius-sm) var(--border-radius-sm) 0;
}

.tip-title {
```

```
  font-weight: 500;
  margin-bottom: var(--spacing-xs);
  color: var(--deep-blue);
}

.tip-text {
  color: var(--deep-blue);
}

.resources-list {
  margin-top: var(--spacing-md);
}

.resource-item {
  margin-bottom: var(--spacing-md);
  display: flex;
  align-items: flex-start;
}

.resource-icon {
  width: 24px;
  height: 24px;
  background-color: var(--soft-lavender);
  border-radius: 50%;
  display: flex;
  align-items: center;
  justify-content: center;
  margin-right: var(--spacing-sm);
  flex-shrink: 0;
}

.resource-content {
  flex: 1;
```

```css
}

.resource-title {
  font-weight: 500;
  margin-bottom: var(--spacing-xs);
}

.resource-description {
  font-size: 0.9rem;
  color: #333;
}

.table-of-contents {
  background-color: #f9f9f9;
  padding: var(--spacing-md);
  border-radius: var(--border-radius-sm);
  margin-bottom: var(--spacing-lg);
}

.toc-title {
  font-weight: 500;
  margin-bottom: var(--spacing-sm);
}

.toc-list {
  list-style-type: none;
  padding-left: var(--spacing-md);
}

.toc-item {
  margin-bottom: var(--spacing-xs);
}
```

```
    .toc-link {
      color: var(--light-blue);
      text-decoration: none;
    }

    .toc-link:hover {
      text-decoration: underline;
    }

    .back-to-top {
      display: block;
      text-align: right;
      margin-top: var(--spacing-md);
      font-size: 0.9rem;
      color: var(--light-blue);
      text-decoration: none;
    }

    .back-to-top:hover {
      text-decoration: underline;
    }
  </style>
</head>
<body>
  <div class="container">
    <div class="header">
      <h1>Crisis Protocols</h1>
      <p>Standard procedures for managing crisis situations in the 988 Lifeline
LGBTQIA+ Crisis Support System</p>
    </div>

    <div class="table-of-contents">
      <div class="toc-title">Table of Contents</div>
```

```
    <ul class="toc-list">
      <li class="toc-item"><a href="#suicide-risk" class="toc-link">Suicide Risk Assessment</a></li>
      <li class="toc-item"><a href="#active-rescue" class="toc-link">Active Rescue Procedure</a></li>
      <li class="toc-item"><a href="#lgbtqia-specific" class="toc-link">LGBTQIA+ Specific Considerations</a></li>
      <li class="toc-item"><a href="#safety-planning" class="toc-link">Safety Planning</a></li>
      <li class="toc-item"><a href="#resources" class="toc-link">Crisis Resources</a></li>
    </ul>
  </div>

  <div class="content">
    <div id="suicide-risk" class="protocol-section">
      <h2 class="protocol-title">Suicide Risk Assessment</h2>

      <div class="protocol-description">
        <p>The suicide risk assessment protocol helps counselors evaluate callers' immediate risk level and determine appropriate interventions. This structured approach ensures consistent evaluation while maintaining a supportive and caring conversation.</p>
      </div>

      <div class="alert-box">
        <div class="alert-title">Important Reminder</div>
        <div class="alert-text">Always prioritize building rapport and connection before direct risk assessment questions. Shift into assessment naturally within the conversation, maintaining empathy throughout.</div>
      </div>

      <div class="steps-list">
```

```
        <div class="step-item">
          <div class="step-title">1. Assess Suicidal Ideation</div>
          <div class="step-description">
            <p>Begin with general questions about thoughts of suicide or
self-harm. Examples:</p>
            <ul>
              <li>"I'm concerned about you. Are you having thoughts of
suicide?"</li>
              <li>"Sometimes when people feel this way, they think about
suicide. Is that something you're thinking about?"</li>
            </ul>
          </div>
        </div>

        <div class="step-item">
          <div class="step-title">2. Evaluate Plan and Means</div>
          <div class="step-description">
            <p>If ideation is present, assess for specific plan and access to
means:</p>
            <ul>
              <li>"Do you have a plan for how you would end your life?"</li>
              <li>"Do you have access to [specific means mentioned]?"</li>
            </ul>
          </div>
        </div>

        <div class="step-item">
          <div class="step-title">3. Determine Timeframe and Intent</div>
          <div class="step-description">
            <p>Evaluate immediacy of risk and strength of intent:</p>
            <ul>
              <li>"When are you thinking about doing this?"</li>
```

```
              <li>"How strong is your desire to act on these thoughts right
now?"</li>
            </ul>
          </div>
        </div>

        <div class="step-item">
          <div class="step-title">4. Identify Protective Factors</div>
          <div class="step-description">
            <p>Explore reasons for living and supports:</p>
            <ul>
              <li>"What has kept you going until now?"</li>
              <li>"Who in your life would you feel comfortable reaching out
to?"</li>
            </ul>
          </div>
        </div>

        <div class="step-item">
          <div class="step-title">5. Determine Risk Level</div>
          <div class="step-description">
            <p>Based on assessment, categorize risk as:</p>
            <ul>
              <li><strong>Low Risk:</strong> Thoughts but no plan, strong
protective factors</li>
              <li><strong>Moderate Risk:</strong> Thoughts with vague plan,
some protective factors</li>
              <li><strong>High Risk:</strong> Specific plan, access to means,
imminent timeframe, few protective factors</li>
            </ul>
          </div>
        </div>
      </div>
```

```
      <div class="tip-box">
        <div class="tip-title">LGBTQIA+ Sensitivity Tip</div>
        <div class="tip-text">When assessing risk with LGBTQIA+ callers, be mindful that experiences of rejection, discrimination, or issues related to gender and sexual identity may be contributing factors. Create space for these specific concerns while avoiding assumptions.</div>
      </div>

      <a href="#" class="back-to-top">Back to Top</a>
    </div>

    <div id="active-rescue" class="protocol-section">
      <h2 class="protocol-title">Active Rescue Procedure</h2>

      <div class="protocol-description">
        <p>The active rescue protocol outlines steps to take when a caller is at imminent risk of suicide and requires emergency intervention. This procedure balances the need for safety with respect for the caller's autonomy when possible.</p>
      </div>

      <div class="alert-box">
        <div class="alert-title">Critical Information</div>
        <div class="alert-text">Active rescue should be initiated when a caller is at imminent risk of suicide with intent, plan, and means, and is unwilling or unable to take steps to ensure their immediate safety.</div>
      </div>

      <div class="steps-list">
        <div class="step-item">
          <div class="step-title">1. Confirm Immediate Risk</div>
          <div class="step-description">
```

```
            <p>Verify that the caller presents with:</p>
            <ul>
              <li>Current suicidal ideation</li>
              <li>Specific and lethal plan</li>
              <li>Access to means</li>
              <li>Intent to act imminently</li>
              <li>Inability to develop a safe alternative</li>
            </ul>
          </div>
        </div>

        <div class="step-item">
          <div class="step-title">2. Attempt Collaborative Safety
Planning</div>
          <div class="step-description">
            <p>Before moving to non-consensual rescue:</p>
            <ul>
              <li>Try to develop a safety plan collaboratively</li>
              <li>Explore removal of means or having someone stay with
them</li>
              <li>Discuss voluntary emergency services engagement</li>
            </ul>
          </div>
        </div>

        <div class="step-item">
          <div class="step-title">3. Gather Location Information</div>
          <div class="step-description">
            <p>If a rescue is necessary, obtain:</p>
            <ul>
              <li>Caller's current location (address, city, state)</li>
              <li>Caller's name</li>
              <li>Description of residence if applicable</li>
```

```
              <li>Phone number they're calling from</li>
            </ul>
            <p>Be transparent about why you need this information.</p>
          </div>
        </div>

        <div class="step-item">
          <div class="step-title">4. Involve Supervisor</div>
          <div class="step-description">
            <p>Alert your supervisor while maintaining the call connection.
They will:</p>
            <ul>
              <li>Confirm risk assessment and rescue necessity</li>
              <li>Contact emergency services</li>
              <li>Document the intervention</li>
            </ul>
          </div>
        </div>

        <div class="step-item">
          <div class="step-title">5. Stay Connected</div>
          <div class="step-description">
            <p>Maintain call connection until:</p>
            <ul>
              <li>Emergency services arrive</li>
              <li>Another support person assumes responsibility</li>
              <li>Risk level significantly decreases</li>
            </ul>
          </div>
        </div>

        <div class="step-item">
          <div class="step-title">6. Document Thoroughly</div>
```

```
            <div class="step-description">
              <p>Complete detailed documentation including:</p>
              <ul>
                <li>Risk assessment details</li>
                <li>Attempts at collaborative safety planning</li>
                <li>Time emergency services were contacted</li>
                <li>Information provided to emergency services</li>
                <li>Outcome of the intervention</li>
              </ul>
            </div>
          </div>
        </div>

        <div class="tip-box">
          <div class="tip-title">LGBTQIA+ Safety Consideration</div>
          <div class="tip-text">Be aware that emergency services interactions can sometimes pose additional risks for LGBTQIA+ individuals, particularly transgender and gender-diverse people. When possible, provide responders with information about appropriate name/pronouns and any specific safety concerns.</div>
        </div>

        <a href="#" class="back-to-top">Back to Top</a>
      </div>

      <div id="lgbtqia-specific" class="protocol-section">
        <h2 class="protocol-title">LGBTQIA+ Specific Considerations</h2>

        <div class="protocol-description">
          <p>This protocol outlines important considerations when supporting LGBTQIA+ individuals in crisis. Understanding these unique factors helps provide culturally responsive care that acknowledges specific stressors and experiences.</p>
```

```
      </div>

      <div class="tip-box">
        <div class="tip-title">Affirmation and Validation</div>
        <div class="tip-text">Always use callers' stated names and pronouns. Affirm their identity as valid and worthy of respect, regardless of others' reactions or treatment. This foundation of respect is essential for effective crisis support.</div>
      </div>

      <div class="steps-list">
        <div class="step-item">
          <div class="step-title">1. Identity-Based Stressors</div>
          <div class="step-description">
            <p>Be aware of specific stressors that may contribute to crisis:</p>
            <ul>
              <li>Family rejection or non-acceptance</li>
              <li>Housing insecurity or homelessness</li>
              <li>Discrimination in employment, healthcare, or education</li>
              <li>Minority stress and internalized negative messages</li>
              <li>Gender dysphoria and transition-related challenges</li>
              <li>Experiences of bullying, harassment, or violence</li>
            </ul>
          </div>
        </div>

        <div class="step-item">
          <div class="step-title">2. Safety Assessment Considerations</div>
          <div class="step-description">
            <p>When assessing safety, include LGBTQIA+-specific factors:</p>
            <ul>
```

```
                <li>Safety of home environment regarding identity
acceptance</li>
                <li>Access to affirming healthcare providers</li>
                <li>Community resources specifically for LGBTQIA+
individuals</li>
                <li>Protective factors within LGBTQIA+ community
connections</li>
                <li>Safety concerns with authorities or emergency services</li>
              </ul>
            </div>
          </div>

          <div class="step-item">
            <div class="step-title">3. Resource Navigation</div>
            <div class="step-description">
              <p>Be prepared to connect callers with specialized resources:</p>
              <ul>
                <li>LGBTQIA+-affirming mental health providers</li>
                <li>Transgender healthcare resources</li>
                <li>LGBTQIA+ youth shelters and housing programs</li>
                <li>Legal services for discrimination issues</li>
                <li>Online and in-person LGBTQIA+ support communities</li>
              </ul>
            </div>
          </div>

          <div class="step-item">
            <div class="step-title">4. Intersectionality Awareness</div>
            <div class="step-description">
              <p>Recognize how multiple identities interact and impact
crisis:</p>
              <ul>
```

```
                <li>Consider racial, cultural, religious, and disability
intersections</li>
                <li>Acknowledge varying access to resources based on
socioeconomic status</li>
                <li>Understand how immigration status may affect service
access</li>
                <li>Respect cultural and religious contexts while supporting
identity</li>
              </ul>
            </div>
          </div>
        </div>

        <div class="alert-box">
          <div class="alert-title">Language and Terminology</div>
          <div class="alert-text">Follow the caller's lead in language about
their identity. If unsure, ask respectfully how they describe their identity
rather than making assumptions. Staying current with terminology is important,
but individual preferences always take precedence.</div>
        </div>

        <a href="#" class="back-to-top">Back to Top</a>
      </div>

      <div id="safety-planning" class="protocol-section">
        <h2 class="protocol-title">Safety Planning</h2>

        <div class="protocol-description">
          <p>Safety planning is a collaborative process to develop concrete
strategies for managing suicidal thoughts and crises. This protocol outlines
key components of effective safety planning that can be tailored to each
caller's unique circumstances.</p>
        </div>
```

```
      <div class="steps-list">
        <div class="step-item">
          <div class="step-title">1. Recognize Warning Signs</div>
          <div class="step-description">
            <p>Help the caller identify personal warning signs of an
impending crisis:</p>
            <ul>
              <li>Thoughts: "What thoughts come up when you start feeling
suicidal?"</li>
              <li>Emotions: "What emotions signal you might be heading toward
crisis?"</li>
              <li>Behaviors: "What do you notice yourself doing
differently?"</li>
              <li>Physical sensations: "What body sensations tell you you're
starting to struggle?"</li>
              <li>Social changes: "How do your interactions with others
change?"</li>
            </ul>
          </div>
        </div>

        <div class="step-item">
          <div class="step-title">2. Internal Coping Strategies</div>
          <div class="step-description">
            <p>Develop self-managed coping techniques:</p>
            <ul>
              <li>Grounding exercises (5-4-3-2-1 senses technique, deep
breathing)</li>
              <li>Physical activities (walking, stretching, exercise)</li>
              <li>Distraction methods (music, creative activities,
games)</li>
```

```
              <li>Emotional regulation strategies (journaling,
meditation)</li>
              <li>Self-compassion practices</li>
            </ul>
          </div>
        </div>

        <div class="step-item">
          <div class="step-title">3. Social Connections and Settings</div>
          <div class="step-description">
            <p>Identify people and places that provide distraction and
comfort:</p>
            <ul>
              <li>Safe social environments (coffee shops, libraries,
community centers)</li>
              <li>People who can provide company without discussing the
crisis</li>
              <li>Structured social activities or groups</li>
              <li>Online communities when in-person options aren't
available</li>
            </ul>
          </div>
        </div>

        <div class="step-item">
          <div class="step-title">4. Contact Support Persons</div>
          <div class="step-description">
            <p>Identify specific people who can help during a crisis:</p>
            <ul>
              <li>Create a prioritized list with contact information</li>
              <li>Include at least 3-5 contacts if possible</li>
              <li>Discuss what the caller feels comfortable sharing with each
person</li>
```

```
              <li>Consider different types of support (emotional support,
practical help)</li>
            </ul>
          </div>
        </div>

        <div class="step-item">
          <div class="step-title">5. Professional Resources</div>
          <div class="step-description">
            <p>Compile crisis services and professional support options:</p>
            <ul>
              <li>Crisis hotlines and text lines (988, Trevor Project)</li>
              <li>Local mental health crisis services</li>
              <li>Mental health providers' crisis protocols</li>
              <li>Nearest emergency departments</li>
              <li>Warm lines for non-emergency support</li>
            </ul>
          </div>
        </div>

        <div class="step-item">
          <div class="step-title">6. Making the Environment Safe</div>
          <div class="step-description">
            <p>Develop strategies to reduce access to means:</p>
            <ul>
              <li>Identify specific lethal means that should be removed or
secured</li>
              <li>Create concrete steps for means removal (who can help,
where items can be stored)</li>
              <li>Consider temporary storage options for dangerous items</li>
              <li>Address prescription medication safety if relevant</li>
            </ul>
          </div>
```

```
        </div>

        <div class="step-item">
          <div class="step-title">7. Reasons for Living</div>
          <div class="step-description">
            <p>Help the caller identify personal motivations to stay
alive:</p>
            <ul>
              <li>Relationships (people, pets)</li>
              <li>Future goals and aspirations</li>
              <li>Values and beliefs</li>
              <li>Previous successes in overcoming difficult times</li>
              <li>Small pleasures and moments of joy</li>
            </ul>
          </div>
        </div>
      </div>

      <div class="tip-box">
        <div class="tip-title">Documentation Tip</div>
        <div class="tip-text">Encourage the caller to document their safety
plan in a format that works for them - written, digital, or as a phone note.
Having the plan accessible during crisis moments is crucial for its
effectiveness.</div>
      </div>

      <a href="#" class="back-to-top">Back to Top</a>
    </div>

    <div id="resources" class="protocol-section">
      <h2 class="protocol-title">Crisis Resources</h2>

      <div class="protocol-description">
```

```
        <p>This section provides an overview of key resources to share with callers in crisis. These resources are organized by type and include LGBTQIA+-specific options when available.</p>
      </div>

      <div class="resources-list">
        <div class="resource-item">
          <div class="resource-icon">📞</div>
          <div class="resource-content">
            <div class="resource-title">National Crisis Hotlines</div>
            <div class="resource-description">
              <ul>
                <li><strong>988 Suicide & Crisis Lifeline:</strong> Call or text 988 (Available 24/7)</li>
                <li><strong>Trevor Project:</strong> 1-866-488-7386 (LGBTQ youth crisis support, 24/7)</li>
                <li><strong>Trans Lifeline:</strong> 1-877-565-8860 (Peer support for trans people)</li>
                <li><strong>Crisis Text Line:</strong> Text HOME to 741741 (24/7 text support)</li>
              </ul>
            </div>
          </div>
        </div>

        <div class="resource-item">
          <div class="resource-icon">🏠</div>
          <div class="resource-content">
            <div class="resource-title">Housing Resources</div>
            <div class="resource-description">
              <ul>
                <li><strong>True Colors United:</strong> Resources for LGBTQ youth experiencing homelessness</li>
```

```
                <li><strong>National Runaway Safeline:</strong> 1-800-RUNAWAY (1-800-786-2929)</li>
                <li><strong>LGBTQ-Affirming Shelters Directory:</strong> Available through findhelp.org</li>
              </ul>
            </div>
          </div>
        </div>

        <div class="resource-item">
          <div class="resource-icon">⚕</div>
          <div class="resource-content">
            <div class="resource-title">Healthcare Resources</div>
            <div class="resource-description">
              <ul>
                <li><strong>GLMA Provider Directory:</strong> LGBTQ-affirming healthcare providers</li>
                <li><strong>National Alliance on Mental Illness (NAMI):</strong> 1-800-950-NAMI (6264)</li>
                <li><strong>RAD Remedy:</strong> Connects trans, gender non-conforming, and intersex people to compassionate healthcare providers</li>
              </ul>
            </div>
          </div>
        </div>

        <div class="resource-item">
          <div class="resource-icon">⚖️</div>
          <div class="resource-content">
            <div class="resource-title">Legal Resources</div>
            <div class="resource-description">
              <ul>
```

```
                <li><strong>Lambda Legal:</strong> Legal support for LGBTQ
people and those living with HIV</li>
                <li><strong>Transgender Law Center:</strong> Legal
information and resources for transgender people</li>
                <li><strong>National Center for Transgender
Equality:</strong> ID document change information</li>
              </ul>
            </div>
          </div>
        </div>

        <div class="resource-item">
          <div class="resource-icon">🌐</div>
          <div class="resource-content">
            <div class="resource-title">Online Communities</div>
            <div class="resource-description">
              <ul>
                <li><strong>Trevor Space:</strong> Online community for LGBTQ
youth ages 13-24</li>
                <li><strong>Q Chat Space:</strong> Live, chat-based support
groups for LGBTQ teens</li>
                <li><strong>Reddit Communities:</strong> r/LGBTeens,
r/asktransgender, r/nonbinary (age-appropriate options)</li>
              </ul>
            </div>
          </div>
        </div>
      </div>

      <div class="alert-box">
        <div class="alert-title">Resource Updates</div>
```

```
        <div class="alert-text">Resources may change over time. Always verify contact information and availability before sharing with callers. Report any outdated information to your supervisor for system-wide updates.</div>
      </div>

      <a href="#" class="back-to-top">Back to Top</a>
    </div>
  </div>
</div>
 <script>
  // Handle "Back to Top" links
  document.querySelectorAll('.back-to-top').forEach(function(link) {
    link.addEventListener('click', function(e) {
      e.preventDefault();
      window.scrollTo({
        top: 0,
        behavior: 'smooth'
      });
    });
  });

  // Handle table of contents links
  document.querySelectorAll('.toc-link').forEach(function(link) {
    link.addEventListener('click', function(e) {
      e.preventDefault();
      const targetId = this.getAttribute('href').substring(1);
      const targetElement = document.getElementById(targetId);

      if (targetElement) {
        window.scrollTo({
          top: targetElement.offsetTop - 20,
          behavior: 'smooth'
        });
```

```
      }
    });
  });
 </script>
</body>
</html>
<!DOCTYPE html>
<html>
<head>
 <base target="_top">
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <link
href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500;700&fam
ily=Poppins:wght@500;600;700&display=swap" rel="stylesheet">
 <style>
   /* Variables for 988 Lifeline LGBTQIA+ Services color scheme */
   :root {
     /* Primary Colors */
     --trevor-orange: #FF786E;
     --deep-blue: #001A4E;
     --purple: #9A3499;
     --teal: #137F6A;

     /* Secondary Colors */
     --light-blue: #4F52DE;
     --soft-yellow: #FFAD8D;
     --lavender: #B3AE4A;
     --light-purple: #F54AC;

     /* Tertiary Colors */
     --soft-green: #BAE2CE;
     --pale-yellow: #FFF2DF;
```

```
    --light-pink: #FBCBBE;
    --soft-lavender: #D1CFCC;

    /* Gradients */
    --primary-gradient: linear-gradient(135deg, var(--trevor-orange),
var(--light-pink));
    --calm-gradient: linear-gradient(135deg, var(--light-blue),
var(--soft-green));
    --support-gradient: linear-gradient(45deg, var(--purple),
var(--light-purple));
    --background-gradient: linear-gradient(180deg, var(--pale-yellow),
#FFFFFF);

    /* Spacing */
    --spacing-xs: 4px;
    --spacing-sm: 8px;
    --spacing-md: 16px;
    --spacing-lg: 24px;
    --spacing-xl: 32px;

    /* Borders */
    --border-radius-sm: 8px;
    --border-radius-md: 12px;
    --border-radius-lg: 16px;
  }

  body {
    font-family: 'Roboto', sans-serif;
    margin: 0;
    padding: 0;
    background-color: var(--pale-yellow);
    color: var(--deep-blue);
  }
```

```css
.container {
  max-width: 600px;
  margin: 0 auto;
  padding: var(--spacing-md);
}

.header {
  margin-bottom: var(--spacing-lg);
  text-align: center;
}

h1, h2, h3, h4 {
  font-family: 'Poppins', sans-serif;
  color: var(--deep-blue);
}

.form-card {
  background: white;
  border-radius: var(--border-radius-md);
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.05);
  padding: var(--spacing-lg);
  position: relative;
}

.form-card::before {
  content: '';
  position: absolute;
  left: 0;
  top: 0;
  height: 100%;
  width: 3px;
  background: var(--purple);
```

```css
}

.form-group {
  margin-bottom: var(--spacing-md);
}

label {
  display: block;
  font-weight: 500;
  margin-bottom: var(--spacing-xs);
}

input, select, textarea {
  width: 100%;
  padding: var(--spacing-sm);
  border: 1px solid #ddd;
  border-radius: var(--border-radius-sm);
  font-family: 'Roboto', sans-serif;
}

.week-preview {
  margin-top: var(--spacing-lg);
  padding: var(--spacing-md);
  background-color: #f9f9f9;
  border-radius: var(--border-radius-sm);
}

.preview-title {
  font-weight: 500;
  margin-bottom: var(--spacing-sm);
}

.days-grid {
```

```
  display: grid;
  grid-template-columns: repeat(7, 1fr);
  gap: var(--spacing-xs);
  margin-bottom: var(--spacing-md);
}

.day-header {
  text-align: center;
  font-weight: 500;
  font-size: 14px;
}

.day-date {
  text-align: center;
  font-size: 12px;
  color: #666;
}

.notes-section {
  margin-top: var(--spacing-sm);
}

.btn {
  border: none;
  padding: var(--spacing-md) var(--spacing-lg);
  border-radius: var(--border-radius-sm);
  font-weight: 500;
  cursor: pointer;
  transition: all 0.2s ease;
}

.btn-primary {
  background: var(--primary-gradient);
```

```
  color: white;
}

.btn-primary:hover {
  box-shadow: 0 4px 8px rgba(255, 120, 110, 0.3);
  transform: translateY(-2px);
}

.btn-secondary {
  background: white;
  color: var(--deep-blue);
  border: 1px solid #ddd;
}

.btn-secondary:hover {
  background: #f5f5f5;
}

.form-actions {
  display: flex;
  justify-content: space-between;
  margin-top: var(--spacing-lg);
}

.success-message {
  background-color: var(--soft-green);
  color: var(--deep-blue);
  padding: var(--spacing-md);
  border-radius: var(--border-radius-sm);
  margin-bottom: var(--spacing-lg);
  text-align: center;
  display: none;
}
```

```
    .error-message {
      background-color: var(--light-pink);
      color: var(--deep-blue);
      padding: var(--spacing-md);
      border-radius: var(--border-radius-sm);
      margin-bottom: var(--spacing-lg);
      text-align: center;
      display: none;
    }

    .counselor-assignment {
      margin-top: var(--spacing-md);
    }

    .assignment-row {
      display: flex;
      align-items: center;
      margin-bottom: var(--spacing-sm);
    }

    .assignment-shift {
      width: 120px;
      font-weight: 500;
    }

    .assignment-select {
      flex: 1;
    }
  </style>
</head>
<body>
  <div class="container">
```

```
  <div class="header">
    <h1>Initialize Week Schedule</h1>
    <p>Set up a standard weekly schedule for the 988 Lifeline LGBTQIA+ Crisis Support Team</p>
  </div>

  <div id="successMessage" class="success-message">
    Week initialized successfully!
  </div>

  <div id="errorMessage" class="error-message">
    An error occurred. Please try again.
  </div>

  <div class="form-card">
    <div class="form-group">
      <label for="weekStartDate">Week Starting</label>
      <input type="date" id="weekStartDate" onchange="updatePreview()">
    </div>

    <div class="form-group">
      <label>Select Template</label>
      <div>
        <input type="radio" id="templateStandard" name="template" value="standard" checked onchange="updateAssignmentOptions()">
        <label for="templateStandard" style="display: inline;">Standard Week</label>

        <input type="radio" id="templateCustom" name="template" value="custom" style="margin-left: 20px;" onchange="updateAssignmentOptions()">
        <label for="templateCustom" style="display: inline;">Custom Assignments</label>
```

```
      </div>
    </div>

    <div id="counselorAssignments" class="counselor-assignment"
style="display: none;">
      <h3>Shift Assignments</h3>

      <!-- Morning Shift Assignments -->
      <div class="assignment-row">
        <div class="assignment-shift">Morning</div>
        <div class="assignment-select">
          <select id="mondayMorning">
            <option value="">-- Select Counselor --</option>
            <!-- Counselors will be populated here -->
          </select>
        </div>
      </div>

      <div class="assignment-row">
        <div class="assignment-shift">Afternoon</div>
        <div class="assignment-select">
          <select id="mondayAfternoon">
            <option value="">-- Select Counselor --</option>
            <!-- Counselors will be populated here -->
          </select>
        </div>
      </div>

      <div class="assignment-row">
        <div class="assignment-shift">Night</div>
        <div class="assignment-select">
          <select id="mondayNight">
            <option value="">-- Select Counselor --</option>
```

```
            <!-- Counselors will be populated here -->
          </select>
        </div>
      </div>
    </div>

    <div class="week-preview">
      <div class="preview-title">Week Preview</div>
      <div class="days-grid">
        <div class="day-header">Sun</div>
        <div class="day-header">Mon</div>
        <div class="day-header">Tue</div>
        <div class="day-header">Wed</div>
        <div class="day-header">Thu</div>
        <div class="day-header">Fri</div>
        <div class="day-header">Sat</div>

        <div class="day-date" id="preview-sun">3/9</div>
        <div class="day-date" id="preview-mon">3/10</div>
        <div class="day-date" id="preview-tue">3/11</div>
        <div class="day-date" id="preview-wed">3/12</div>
        <div class="day-date" id="preview-thu">3/13</div>
        <div class="day-date" id="preview-fri">3/14</div>
        <div class="day-date" id="preview-sat">3/15</div>
      </div>

      <div class="notes-section">
        <p>This will create a template schedule for the selected week. Each
shift will be populated with the assigned counselors according to the
template.</p>
        <p>You can make individual adjustments later in the Schedule
Manager.</p>
      </div>
```

```
    </div>

    <div class="form-actions">
      <button type="button" class="btn btn-secondary"
onclick="cancelForm()">Cancel</button>
      <button type="button" class="btn btn-primary"
onclick="initializeWeek()">Initialize Week</button>
    </div>
  </div>
</div>
 <script>
  // Initialize the form
  function initialize() {
    // Set default week start date to next Sunday
    setNextSunday();

    // Update the preview dates
    updatePreview();

    // Load counselors
    loadCounselors();
  }

  // Set the date input to next Sunday
  function setNextSunday() {
    const today = new Date();
    const dayOfWeek = today.getDay(); // 0 = Sunday, 6 = Saturday

    // Calculate days until next Sunday
    const daysUntilNextSunday = (7 - dayOfWeek) % 7;

    // If today is Sunday, use next Sunday
    const daysToAdd = daysUntilNextSunday === 0 ? 7 : daysUntilNextSunday;
```

```
    // Create next Sunday date
    const nextSunday = new Date(today);
    nextSunday.setDate(today.getDate() + daysToAdd);

    // Set the date input
    document.getElementById('weekStartDate').value =
nextSunday.toISOString().split('T')[0];
  }

  // Update the preview dates based on selected start date
  function updatePreview() {
    const startDateInput = document.getElementById('weekStartDate');

    if (!startDateInput.value) {
      return;
    }

    const startDate = new Date(startDateInput.value);
    const days = ['sun', 'mon', 'tue', 'wed', 'thu', 'fri', 'sat'];

    days.forEach((day, index) => {
      const date = new Date(startDate);
      date.setDate(startDate.getDate() + index);

      // Format as M/D
      const formattedDate = `${date.getMonth() + 1}/${date.getDate()}`;
      document.getElementById(`preview-${day}`).textContent = formattedDate;
    });
  }

  // Update assignment options based on template selection
  function updateAssignmentOptions() {
```

```
    const templateCustom = document.getElementById('templateCustom').checked;
    document.getElementById('counselorAssignments').style.display =
templateCustom ? 'block' : 'none';
  }

  // Load counselors
  function loadCounselors() {
    google.script.run
      .withSuccessHandler(function(counselors) {
        populateCounselorSelects(counselors);
      })
      .withFailureHandler(function(error) {
        showError(`Failed to load counselors: ${error}`);
      })
      .getTeamMembers();
  }

  // Populate counselor select dropdowns
  function populateCounselorSelects(counselors) {
    const selects = [
      document.getElementById('mondayMorning'),
      document.getElementById('mondayAfternoon'),
      document.getElementById('mondayNight')
    ];

    selects.forEach(select => {
      // Keep the first option (empty)
      while (select.options.length > 1) {
        select.remove(1);
      }

      // Add counselors
      counselors.forEach(counselor => {
```

```
        const option = document.createElement('option');
        option.value = counselor.email;
        option.textContent = counselor.name;
        select.appendChild(option);
      });
    });
  }

  // Initialize week
  function initializeWeek() {
    // Get week start date
    const startDate = document.getElementById('weekStartDate').value;

    if (!startDate) {
      showError('Please select a week start date');
      return;
    }

    // Get template type
    const templateType =
document.querySelector('input[name="template"]:checked').value;

    // Get counselor assignments if using custom template
    let counselorAssignments = null;

    if (templateType === 'custom') {
      counselorAssignments = {
        mondayMorning: document.getElementById('mondayMorning').value,
        mondayAfternoon: document.getElementById('mondayAfternoon').value,
        mondayNight: document.getElementById('mondayNight').value
      };
    }
```

```
    // Create data object
    const weekData = {
      startDate: startDate,
      templateType: templateType,
      counselorAssignments: counselorAssignments
    };

    // Submit to server
    google.script.run
      .withSuccessHandler(function(result) {
        if (result.success) {
          showSuccess(result.message || 'Week initialized successfully!');

          // Reset form
          setNextSunday();
          updatePreview();
          document.getElementById('templateStandard').checked = true;
          updateAssignmentOptions();
        } else {
          showError(result.message || 'Failed to initialize week');
        }
      })
      .withFailureHandler(function(error) {
        showError(`Error initializing week: ${error}`);
      })
      .initializeWeek(weekData);
  }

  // Cancel form
  function cancelForm() {
    google.script.host.close();
  }
```

```
    // Show success message
    function showSuccess(message) {
      const successElement = document.getElementById('successMessage');
      successElement.textContent = message;
      successElement.style.display = 'block';

      // Hide after 3 seconds
      setTimeout(() => {
        successElement.style.display = 'none';
      }, 3000);
    }

    // Show error message
    function showError(message) {
      const errorElement = document.getElementById('errorMessage');
      errorElement.textContent = message;
      errorElement.style.display = 'block';

      // Hide after 5 seconds
      setTimeout(() => {
        errorElement.style.display = 'none';
      }, 5000);
    }

    // Initialize when the page loads
    google.script.onload = function() {
      initialize();
    };
  </script>
</body>
</html>
<!DOCTYPE html>
<html>
```

```
<head>
 <base target="_top">
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <link
href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500;700&fam
ily=Poppins:wght@500;600;700&display=swap" rel="stylesheet">
 <style>
   /* Variables for Trevor Project color scheme */
   :root {
     /* Primary Colors */
     --trevor-orange: #FF786E;
     --deep-blue: #001A4E;
     --purple: #9A3499;
     --teal: #137F6A;

     /* Secondary Colors */
     --light-blue: #4F52DE;
     --soft-yellow: #FFAD8D;
     --lavender: #B3AE4A;
     --light-purple: #F54AC;

     /* Tertiary Colors */
     --soft-green: #BAE2CE;
     --pale-yellow: #FFF2DF;
     --light-pink: #FBCBBE;
     --soft-lavender: #D1CFCC;

     /* Gradients */
     --primary-gradient: linear-gradient(135deg, var(--trevor-orange),
var(--light-pink));
     --calm-gradient: linear-gradient(135deg, var(--light-blue),
var(--soft-green));
```

```
    --support-gradient: linear-gradient(45deg, var(--purple),
var(--light-purple));
    --background-gradient: linear-gradient(180deg, var(--pale-yellow),
#FFFFFF);

    /* Spacing */
    --spacing-xs: 4px;
    --spacing-sm: 8px;
    --spacing-md: 16px;
    --spacing-lg: 24px;
    --spacing-xl: 32px;

    /* Border Radius */
    --border-radius-sm: 8px;
    --border-radius-md: 12px;
    --border-radius-lg: 16px;
  }

  body {
    font-family: 'Roboto', sans-serif;
    margin: 0;
    padding: 0;
    background: var(--background-gradient);
    color: var(--deep-blue);
    min-height: 100vh;
  }

  h1, h2, h3, h4, h5, h6 {
    font-family: 'Poppins', sans-serif;
    color: var(--deep-blue);
  }

  .container {
```

```
  max-width: 1200px;
  margin: 0 auto;
  padding: var(--spacing-md);
}

.header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  margin-bottom: var(--spacing-lg);
  padding-bottom: var(--spacing-md);
  border-bottom: 1px solid rgba(0, 0, 0, 0.1);
}

.header-actions {
  display: flex;
  gap: var(--spacing-sm);
}

.card {
  background: white;
  border-radius: var(--border-radius-md);
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.05);
  margin-bottom: var(--spacing-lg);
  overflow: hidden;
  position: relative;
}

.card::before {
  content: '';
  position: absolute;
  left: 0;
  top: 0;
```

```
  height: 100%;
  width: 3px;
  background: var(--trevor-orange);
}

.card-header {
  padding: var(--spacing-md);
  background: rgba(255, 120, 110, 0.05);
  display: flex;
  justify-content: space-between;
  align-items: center;
}

.card-content {
  padding: var(--spacing-md);
}

.form-grid {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(250px, 1fr));
  gap: var(--spacing-md);
  margin-bottom: var(--spacing-lg);
}

.form-group {
  margin-bottom: var(--spacing-md);
}

.form-label {
  display: block;
  margin-bottom: var(--spacing-xs);
  font-weight: 500;
}
```

```css
.form-control {
  width: 100%;
  padding: var(--spacing-sm);
  border: 1px solid #ddd;
  border-radius: var(--border-radius-sm);
  font-family: 'Roboto', sans-serif;
}

.form-control:focus {
  outline: none;
  border-color: var(--trevor-orange);
  box-shadow: 0 0 0 2px rgba(255, 120, 110, 0.2);
}

textarea.form-control {
  min-height: 100px;
  resize: vertical;
}

.form-section {
  margin-bottom: var(--spacing-lg);
  padding-bottom: var(--spacing-md);
  border-bottom: 1px solid #eee;
}

.form-section:last-child {
  border-bottom: none;
}

.section-title {
  display: flex;
  align-items: center;
```

```
  margin-bottom: var(--spacing-md);
}

.section-icon {
  width: 24px;
  height: 24px;
  display: flex;
  align-items: center;
  justify-content: center;
  background: rgba(255, 120, 110, 0.1);
  border-radius: 50%;
  margin-right: var(--spacing-sm);
}

.btn {
  border: none;
  padding: var(--spacing-sm) var(--spacing-md);
  border-radius: var(--border-radius-sm);
  font-weight: 500;
  cursor: pointer;
  transition: all 0.2s ease;
}

.btn-primary {
  background: var(--primary-gradient);
  color: white;
}

.btn-primary:hover {
  box-shadow: 0 4px 8px rgba(255, 120, 110, 0.3);
  transform: translateY(-2px);
}
```

```css
.btn-secondary {
  background: white;
  color: var(--deep-blue);
  border: 1px solid #ddd;
}

.btn-secondary:hover {
  background: #f5f5f5;
}

.meetings-list {
  margin-top: var(--spacing-lg);
}

.meeting-item {
  background: white;
  border-radius: var(--border-radius-sm);
  padding: var(--spacing-md);
  margin-bottom: var(--spacing-md);
  display: flex;
  justify-content: space-between;
  align-items: center;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.05);
}

.meeting-info {
  flex: 1;
}

.meeting-date {
  font-weight: 500;
  margin-bottom: var(--spacing-xs);
}
```

```css
.meeting-meta {
  font-size: 14px;
  color: #666;
}

.meeting-actions {
  display: flex;
  gap: var(--spacing-sm);
}

.tab-container {
  margin-bottom: var(--spacing-lg);
}

.tabs {
  display: flex;
  border-bottom: 1px solid #eee;
  margin-bottom: var(--spacing-lg);
}

.tab {
  padding: var(--spacing-sm) var(--spacing-md);
  cursor: pointer;
  border-bottom: 3px solid transparent;
  font-weight: 500;
  transition: all 0.2s ease;
}

.tab.active {
  border-bottom-color: var(--trevor-orange);
  color: var(--trevor-orange);
}
```

```css
.tab-content {
  display: none;
}

.tab-content.active {
  display: block;
}

.loading {
  text-align: center;
  padding: var(--spacing-xl);
  color: #666;
}

.empty-state {
  text-align: center;
  padding: var(--spacing-xl);
  color: #666;
}

.empty-state-icon {
  font-size: 48px;
  margin-bottom: var(--spacing-md);
  color: #ddd;
}

.status-badge {
  display: inline-block;
  padding: 2px 8px;
  border-radius: 50px;
  font-size: 12px;
  font-weight: 500;
```

```
  color: white;
}

.status-badge.completed {
  background-color: var(--teal);
}

.status-badge.pending {
  background-color: var(--soft-yellow);
}

.status-badge.overdue {
  background-color: var(--light-purple);
}

/* Footer */
.footer {
  display: flex;
  justify-content: space-between;
  margin-top: var(--spacing-xl);
  padding-top: var(--spacing-lg);
  border-top: 1px solid rgba(0, 0, 0, 0.1);
}

/* Animation */
@keyframes fadeIn {
  0% { opacity: 0; }
  100% { opacity: 1; }
}

.fade-in {
  animation: fadeIn 0.3s ease-in-out;
}
```

```
  /* Modal styles */
  .modal-overlay {
    position: fixed;
    top: 0;
    left: 0;
    right: 0;
    bottom: 0;
    background: rgba(0, 0, 0, 0.5);
    display: flex;
    align-items: center;
    justify-content: center;
    z-index: 1000;
  }

  .modal-content {
    background: white;
    width: 90%;
    max-width: 800px;
    max-height: 90vh;
    overflow-y: auto;
    border-radius: var(--border-radius-md);
    box-shadow: 0 4px 20px rgba(0, 0, 0, 0.15);
  }
 </style>
</head>
<body>
 <div class="container">
   <div class="header">
     <h1>Manager 1:1 Documentation</h1>
     <div class="header-actions">
       <button id="newMeetingBtn" class="btn btn-primary">New 1:1
Meeting</button>
```

```
    </div>
  </div>

  <div class="tab-container">
    <div class="tabs">
      <div class="tab active" data-tab="upcoming">Upcoming & Recent</div>
      <div class="tab" data-tab="history">Meeting History</div>
      <div class="tab" data-tab="actionItems">Action Items</div>
    </div>

    <div id="upcoming-tab" class="tab-content active">
      <!-- New 1:1 Form Card -->
      <div id="newMeetingForm" class="card" style="display: none;">
        <div class="card-header">
          <h3>New Manager 1:1 Meeting</h3>
        </div>
        <div class="card-content">
          <div class="form-grid">
            <div class="form-group">
              <label class="form-label" for="meetingDate">Meeting
Date</label>
              <input type="date" id="meetingDate" class="form-control"
required>
            </div>

            <div class="form-group">
              <label class="form-label" for="nextCheckInDate">Next Check-In
Date</label>
              <input type="date" id="nextCheckInDate" class="form-control">
            </div>

            <div class="form-group">
```

```
              <label class="form-label" for="managerName">Manager
Name</label>
              <input type="text" id="managerName" class="form-control"
required>
            </div>
          </div>

          <div class="form-section">
            <div class="section-title">
              <div class="section-icon">😊</div>
              <h3>Personal Check-In</h3>
            </div>

            <div class="form-group">
              <label class="form-label" for="feelingOverall">How are you
feeling overall?</label>
              <textarea id="feelingOverall" class="form-control"
placeholder="Discuss your overall well-being and energy level..."></textarea>
            </div>

            <div class="form-group">
              <label class="form-label" for="workLifeBalance">Work-Life
Balance</label>
              <textarea id="workLifeBalance" class="form-control"
placeholder="Discuss your current work-life balance..."></textarea>
            </div>
          </div>

          <div class="form-section">
            <div class="section-title">
              <div class="section-icon">🏆</div>
              <h3>Highlights and Wins</h3>
            </div>
```

```
            <div class="form-group">
              <label class="form-label" for="accomplishments">Recent
Accomplishments</label>
              <textarea id="accomplishments" class="form-control"
placeholder="What have you accomplished since our last 1:1?"></textarea>
            </div>

            <div class="form-group">
              <label class="form-label" for="feedbackReflection">Feedback
Reflection</label>
              <textarea id="feedbackReflection" class="form-control"
placeholder="Reflection on any recent feedback received..."></textarea>
            </div>
          </div>

          <div class="form-section">
            <div class="section-title">
              <div class="section-icon">🚧</div>
              <h3>Challenges and Blockers</h3>
            </div>

            <div class="form-group">
              <label class="form-label" for="recentChallenges">Recent
Challenges</label>
              <textarea id="recentChallenges" class="form-control"
placeholder="What challenges have you faced with your team?"></textarea>
            </div>

            <div class="form-group">
              <label class="form-label" for="blockers">Current
Blockers</label>
```

```
              <textarea id="blockers" class="form-control" placeholder="What is preventing you from being effective?"></textarea>
            </div>
          </div>

          <div class="form-section">
            <div class="section-title">
              <div class="section-icon">🌱</div>
              <h3>Growth and Development</h3>
            </div>

            <div class="form-group">
              <label class="form-label" for="growthAreas">Growth Areas</label>
              <textarea id="growthAreas" class="form-control" placeholder="What areas would you like to develop?"></textarea>
            </div>

            <div class="form-group">
              <label class="form-label" for="supportNeeded">Support Needed</label>
              <textarea id="supportNeeded" class="form-control" placeholder="What support do you need from your manager?"></textarea>
            </div>
          </div>

          <div class="form-section">
            <div class="section-title">
              <div class="section-icon">✅</div>
              <h3>Action Items</h3>
            </div>

            <div id="actionItemsContainer">
```

```
              <div class="action-item">
                <div class="form-group">
                  <label class="form-label" for="actionItem1">Action Item
1</label>
                  <input type="text" id="actionItem1" class="form-control"
placeholder="Describe the action item...">
                </div>

                <div class="form-group">
                  <label class="form-label" for="actionItemDueDate1">Due
Date</label>
                  <input type="date" id="actionItemDueDate1"
class="form-control">
                </div>
              </div>

              <button type="button" id="addActionItemBtn" class="btn
btn-secondary" style="margin-top: 10px;">+ Add Another Action Item</button>
            </div>
          </div>
        </div>

        <div class="footer">
          <button type="button" id="cancelMeetingBtn" class="btn
btn-secondary">Cancel</button>
          <button type="button" id="saveMeetingBtn" class="btn
btn-primary">Save Meeting Notes</button>
        </div>
      </div>

      <!-- Upcoming & Recent Meetings -->
      <div id="upcomingMeetings" class="meetings-list">
        <div class="loading" id="loadingIndicator">
```

```
          Loading your meetings...
        </div>

        <div class="empty-state" id="emptyStateUpcoming" style="display:
none;">
          <div class="empty-state-icon">📅</div>
          <h3>No Upcoming Meetings</h3>
          <p>You don't have any upcoming or recent 1:1 meetings with your
manager.</p>
          <button class="btn btn-primary" id="emptyStateNewBtn">Schedule a
1:1 Meeting</button>
        </div>

        <!-- Meeting items will be populated here -->
      </div>
    </div>

    <div id="history-tab" class="tab-content">
      <div class="card">
        <div class="card-header">
          <h3>Meeting History</h3>
          <div class="header-actions">
            <input type="search" id="searchMeetings" class="form-control"
placeholder="Search meetings..." style="max-width: 200px;">
          </div>
        </div>
        <div class="card-content">
          <div id="meetingHistory" class="meetings-list">
            <div class="loading" id="loadingHistoryIndicator">
              Loading meeting history...
            </div>
```

```
            <div class="empty-state" id="emptyStateHistory" style="display:
none;">
              <div class="empty-state-icon">📝</div>
              <h3>No Meeting History</h3>
              <p>You haven't had any 1:1 meetings with your manager yet.</p>
            </div>

            <!-- Meeting history items will be populated here -->
          </div>
        </div>
      </div>

      <div id="actionItems-tab" class="tab-content">
        <div class="card">
          <div class="card-header">
            <h3>Action Items</h3>
            <div class="header-actions">
              <select id="filterActionItems" class="form-control">
                <option value="all">All Status</option>
                <option value="pending">Pending</option>
                <option value="completed">Completed</option>
                <option value="overdue">Overdue</option>
              </select>
            </div>
          </div>
          <div class="card-content">
            <div id="actionItemsList">
              <div class="loading" id="loadingActionsIndicator">
                Loading action items...
              </div>
```

```
            <div class="empty-state" id="emptyStateActions" style="display:
none;">
              <div class="empty-state-icon">✓</div>
              <h3>No Action Items</h3>
              <p>You don't have any action items from your manager 1:1
meetings.</p>
            </div>

            <!-- Action items will be populated here -->
          </div>
        </div>
      </div>
    </div>
  </div>
  <!-- Modal for Meeting Details -->
  <div id="meetingDetailModal" class="modal-overlay" style="display: none;">
    <div class="modal-content card meeting-detail">
      <div class="card-header">
        <h3>1:1 Meeting Details</h3>
        <button class="btn btn-secondary close-modal-btn">Close</button>
      </div>
      <div class="card-content">
        <div class="meeting-meta-info">
          <p><strong>Meeting Date:</strong> <span
id="detailMeetingDate"></span></p>
          <p><strong>Manager:</strong> <span id="detailManagerName"></span></p>
          <p><strong>Next Check-In:</strong> <span
id="detailNextCheckin"></span></p>
        </div>

        <div class="form-section">
          <div class="section-title">
```

```
        <div class="section-icon">😊</div>
        <h3>Personal Check-In</h3>
      </div>
      <p id="detailFeelingOverall"></p>
      <p id="detailWorkLifeBalance"></p>
    </div>

    <div class="form-section">
      <div class="section-title">
        <div class="section-icon">🏆</div>
        <h3>Highlights and Wins</h3>
      </div>
      <p id="detailAccomplishments"></p>
      <p id="detailFeedbackReflection"></p>
    </div>

    <div class="form-section">
      <div class="section-title">
        <div class="section-icon">🚧</div>
        <h3>Challenges and Blockers</h3>
      </div>
      <p id="detailRecentChallenges"></p>
      <p id="detailBlockers"></p>
    </div>

    <div class="form-section">
      <div class="section-title">
        <div class="section-icon">🌱</div>
        <h3>Growth and Development</h3>
      </div>
      <p id="detailGrowthAreas"></p>
      <p id="detailSupportNeeded"></p>
    </div>
```

```
      <div class="form-section">
        <div class="section-title">
          <div class="section-icon">✅</div>
          <h3>Action Items</h3>
        </div>
        <div id="detailActionItems"></div>
      </div>
    </div>

    <div class="footer">
      <button class="btn btn-secondary close-modal-btn">Close</button>
      <button class="btn btn-primary" id="editMeetingBtn">Edit
Meeting</button>
    </div>
  </div>
 </div>
  <script>
   // Current user email
   let currentUserEmail = '';

   // Action item counter for dynamic form elements
   let actionItemCount = 1;

   // Current meetings data
   let meetings = [];
   let actionItems = [];

   // Currently selected meeting ID for details view
   let selectedMeetingId = null;

   // Initialize on page load
   document.addEventListener('DOMContentLoaded', function() {
```

```
    // Load current user
    google.script.run
      .withSuccessHandler(function(email) {
        currentUserEmail = email;
        loadMeetings();
      })
      .withFailureHandler(handleError)
      .getCurrentUserEmail();

    // Set up tab switching
    document.querySelectorAll('.tab').forEach(tab => {
      tab.addEventListener('click', function() {
        const tabId = this.getAttribute('data-tab');
        switchTab(tabId);
      });
    });

    // Set up form buttons
    document.getElementById('newMeetingBtn').addEventListener('click',
showNewMeetingForm);
    document.getElementById('cancelMeetingBtn').addEventListener('click',
hideNewMeetingForm);
    document.getElementById('saveMeetingBtn').addEventListener('click',
saveMeeting);
    document.getElementById('addActionItemBtn').addEventListener('click',
addActionItemField);
    document.getElementById('emptyStateNewBtn').addEventListener('click',
showNewMeetingForm);

    // Set up filter for action items
    document.getElementById('filterActionItems').addEventListener('change',
filterActionItems);
```

```
    // Set up search for meeting history
    document.getElementById('searchMeetings').addEventListener('input',
searchMeetings);

    // Set up modal close buttons
    document.querySelectorAll('.close-modal-btn').forEach(btn => {
      btn.addEventListener('click', closeDetailModal);
    });

    // Set up edit meeting button
    document.getElementById('editMeetingBtn').addEventListener('click',
function() {
      closeDetailModal();
      if (selectedMeetingId) {
        editMeeting(selectedMeetingId);
      }
    });
  });

  // Switch between tabs
  function switchTab(tabId) {
    // Hide all tabs and remove active class
    document.querySelectorAll('.tab-content').forEach(content => {
      content.classList.remove('active');
    });

    document.querySelectorAll('.tab').forEach(tab => {
      tab.classList.remove('active');
    });

    // Show selected tab and add active class
    document.getElementById(`${tabId}-tab`).classList.add('active');
```

```
document.querySelector(`.tab[data-tab="${tabId}"]`).classList.add('active');

    // Load data based on tab
    if (tabId === 'history' &&
document.getElementById('meetingHistory').children.length <= 2) {
      loadMeetingHistory();
    } else if (tabId === 'actionItems' &&
document.getElementById('actionItemsList').children.length <= 2) {
      loadActionItems();
    }
  }

  // Show new meeting form
  function showNewMeetingForm() {
    // Set today's date as default
    const today = new Date().toISOString().split('T')[0];
    document.getElementById('meetingDate').value = today;

    // Reset form fields
    document.getElementById('nextCheckInDate').value = '';
    document.getElementById('managerName').value = '';
    document.getElementById('feelingOverall').value = '';
    document.getElementById('workLifeBalance').value = '';
    document.getElementById('accomplishments').value = '';
    document.getElementById('feedbackReflection').value = '';
    document.getElementById('recentChallenges').value = '';
    document.getElementById('blockers').value = '';
    document.getElementById('growthAreas').value = '';
    document.getElementById('supportNeeded').value = '';

    // Reset action items
    document.getElementById('actionItemsContainer').innerHTML = `
```

```
      <div class="action-item">
        <div class="form-group">
          <label class="form-label" for="actionItem1">Action Item 1</label>
          <input type="text" id="actionItem1" class="form-control"
placeholder="Describe the action item...">
        </div>
        <div class="form-group">
          <label class="form-label" for="actionItemDueDate1">Due Date</label>
          <input type="date" id="actionItemDueDate1" class="form-control">
        </div>
      </div>
      <button type="button" id="addActionItemBtn" class="btn btn-secondary"
style="margin-top: 10px;">+ Add Another Action Item</button>
    `;
    actionItemCount = 1;

    // Show the form with animation
    const form = document.getElementById('newMeetingForm');
    form.style.display = 'block';
    form.classList.add('fade-in');

    // Hide the meetings list
    document.getElementById('upcomingMeetings').style.display = 'none';
  }

  // Hide new meeting form
  function hideNewMeetingForm() {
    document.getElementById('newMeetingForm').style.display = 'none';
    document.getElementById('upcomingMeetings').style.display = 'block';
  }

  // Add another action item field
  function addActionItemField() {
```

```
    actionItemCount++;

    const newActionItem = document.createElement('div');
    newActionItem.className = 'action-item';
    newActionItem.innerHTML = `
      <div class="form-group">
        <label class="form-label" for="actionItem${actionItemCount}">Action
Item ${actionItemCount}</label>
        <input type="text" id="actionItem${actionItemCount}"
class="form-control" placeholder="Describe the action item...">
      </div>
      <div class="form-group">
        <label class="form-label"
for="actionItemDueDate${actionItemCount}">Due Date</label>
        <input type="date" id="actionItemDueDate${actionItemCount}"
class="form-control">
      </div>
      <button type="button" class="btn btn-secondary remove-item-btn"
style="margin-top: 10px;" onclick="removeActionItem(this)">Remove</button>
    `;

    const container = document.getElementById('actionItemsContainer');
    container.insertBefore(newActionItem,
document.getElementById('addActionItemBtn'));
  }

  // Remove an action item field
  function removeActionItem(button) {
    const actionItem = button.parentNode;
    actionItem.parentNode.removeChild(actionItem);
  }

  // Save meeting notes
```

```
  function saveMeeting() {
    // Validate required fields
    const meetingDate = document.getElementById('meetingDate').value;
    const managerName = document.getElementById('managerName').value;

    if (!meetingDate || !managerName) {
      alert('Please fill in all required fields (Meeting Date and Manager Name)');
      return;
    }

    // Collect action items
    const actionItems = [];
    const actionItemElements = document.querySelectorAll('.action-item');

    actionItemElements.forEach((item, index) => {
      const actionItem = item.querySelector(`#actionItem${index + 1}`);
      const actionItemDueDate = item.querySelector(`#actionItemDueDate${index + 1}`);

      if (actionItem && actionItem.value.trim()) {
        actionItems.push({
          description: actionItem.value.trim(),
          dueDate: actionItemDueDate ? actionItemDueDate.value : '',
          status: 'pending'
        });
      }
    });

    // Collect meeting data
    const meetingData = {
      meetingDate: meetingDate,
      nextCheckInDate: document.getElementById('nextCheckInDate').value,
```

```
        managerName: managerName,
        feelingOverall: document.getElementById('feelingOverall').value,
        workLifeBalance: document.getElementById('workLifeBalance').value,
        accomplishments: document.getElementById('accomplishments').value,
        feedbackReflection:
document.getElementById('feedbackReflection').value,
        recentChallenges: document.getElementById('recentChallenges').value,
        blockers: document.getElementById('blockers').value,
        growthAreas: document.getElementById('growthAreas').value,
        supportNeeded: document.getElementById('supportNeeded').value,
        actionItems: actionItems
      };

      // Disable the save button to prevent multiple submissions
      document.getElementById('saveMeetingBtn').disabled = true;
      document.getElementById('saveMeetingBtn').innerText = 'Saving...';

      // Save the meeting
      google.script.run
        .withSuccessHandler(function(result) {
          if (result.success) {
            // Hide the form and reload meetings
            hideNewMeetingForm();
            loadMeetings();
          } else {
            alert('Error saving meeting: ' + result.message);
            document.getElementById('saveMeetingBtn').disabled = false;
            document.getElementById('saveMeetingBtn').innerText = 'Save Meeting
Notes';
          }
        })
        .withFailureHandler(function(error) {
          alert('Error: ' + error.message);
```

```
          document.getElementById('saveMeetingBtn').disabled = false;
          document.getElementById('saveMeetingBtn').innerText = 'Save Meeting
Notes';
        })
        .saveManagerOneOnOne(meetingData);
    }

    // Load recent and upcoming meetings
    function loadMeetings() {
      document.getElementById('loadingIndicator').style.display = 'block';
      document.getElementById('emptyStateUpcoming').style.display = 'none';

      // Remove any existing meeting items
      const meetingsList = document.getElementById('upcomingMeetings');

Array.from(meetingsList.getElementsByClassName('meeting-item')).forEach(item
=> {
        meetingsList.removeChild(item);
      });

      google.script.run
        .withSuccessHandler(function(data) {
          meetings = data;
          document.getElementById('loadingIndicator').style.display = 'none';

          if (data.length === 0) {
            document.getElementById('emptyStateUpcoming').style.display =
'block';
            return;
          }

          // Display recent and upcoming meetings
          data.forEach(meeting => {
```

```
            const meetingItem = createMeetingItem(meeting);
            meetingsList.appendChild(meetingItem);
          });
        })
        .withFailureHandler(handleError)
        .getRecentAndUpcomingManagerOneOnOnes();
    }

    // Load meeting history
    function loadMeetingHistory() {
      document.getElementById('loadingHistoryIndicator').style.display =
'block';
      document.getElementById('emptyStateHistory').style.display = 'none';

      // Remove any existing meeting items
      const historyList = document.getElementById('meetingHistory');

Array.from(historyList.getElementsByClassName('meeting-item')).forEach(item =>
{
        historyList.removeChild(item);
      });

      google.script.run
        .withSuccessHandler(function(data) {
          // Add any new meetings that aren't already in our array
          meetings = meetings.concat(data.filter(meeting =>
            !meetings.some(m => m.id === meeting.id)
          ));

          document.getElementById('loadingHistoryIndicator').style.display =
'none';

          if (data.length === 0) {
```

```
          document.getElementById('emptyStateHistory').style.display =
'block';
          return;
        }

        // Display meeting history
        data.forEach(meeting => {
          const meetingItem = createMeetingItem(meeting);
          historyList.appendChild(meetingItem);
        });
      })
      .withFailureHandler(handleError)
      .getManagerOneOnOneHistory();
  }

  // Load action items
  function loadActionItems() {
    document.getElementById('loadingActionsIndicator').style.display =
'block';
    document.getElementById('emptyStateActions').style.display = 'none';

    // Remove any existing action items
    const actionsList = document.getElementById('actionItemsList');

Array.from(actionsList.getElementsByClassName('action-item')).forEach(item =>
{
      actionsList.removeChild(item);
    });

    google.script.run
      .withSuccessHandler(function(data) {
        actionItems = data;
```

```
        document.getElementById('loadingActionsIndicator').style.display =
'none';

        if (data.length === 0) {
          document.getElementById('emptyStateActions').style.display =
'block';
          return;
        }

        // Display action items
        data.forEach(item => {
          const actionItem = createActionItem(item);
          actionsList.appendChild(actionItem);
        });
      })
      .withFailureHandler(handleError)
      .getManagerOneOnOneActionItems();
  }

  // Create a meeting item for display
  function createMeetingItem(meeting) {
    const meetingItem = document.createElement('div');
    meetingItem.className = 'meeting-item';
    meetingItem.setAttribute('data-id', meeting.id);

    // Format dates
    const meetingDate = new Date(meeting.meetingDate);
    const formattedDate = meetingDate.toLocaleDateString();

    let nextCheckIn = 'Not scheduled';
    if (meeting.nextCheckInDate) {
      const nextDate = new Date(meeting.nextCheckInDate);
      nextCheckIn = nextDate.toLocaleDateString();
```

```
    }

    meetingItem.innerHTML = `
      <div class="meeting-info">
        <div class="meeting-date">${formattedDate} with
${meeting.managerName}</div>
        <div class="meeting-meta">Next check-in: ${nextCheckIn}</div>
      </div>
      <div class="meeting-actions">
        <button class="btn btn-secondary view-meeting-btn">View</button>
      </div>
    `;

    // Add event listener for view button
    meetingItem.querySelector('.view-meeting-btn').addEventListener('click',
function() {
      viewMeetingDetail(meeting.id);
    });

    return meetingItem;
  }

  // Create an action item for display
  function createActionItem(item) {
    const actionItem = document.createElement('div');
    actionItem.className = 'action-item meeting-item';
    actionItem.setAttribute('data-id', item.id);
    actionItem.setAttribute('data-status', item.status);

    // Format dates
    let dueDate = 'No due date';
    if (item.dueDate) {
      const date = new Date(item.dueDate);
```

```
      dueDate = date.toLocaleDateString();
    }

    // Determine status display
    let statusClass = 'pending';
    if (item.status === 'completed') {
      statusClass = 'completed';
    } else if (item.status === 'overdue') {
      statusClass = 'overdue';
    }

    actionItem.innerHTML = `
      <div class="meeting-info">
        <div class="meeting-date">${item.description}</div>
        <div class="meeting-meta">
          From meeting on ${new Date(item.meetingDate).toLocaleDateString()}
•
          Due: ${dueDate} •
          <span class="status-badge ${statusClass}">${item.status}</span>
        </div>
      </div>
      <div class="meeting-actions">
        ${item.status !== 'completed' ?
          '<button class="btn btn-secondary complete-action-btn">Mark
Complete</button>' :
          '<button class="btn btn-secondary
reopen-action-btn">Reopen</button>'}
      </div>
    `;

    // Add event listeners for buttons
    if (item.status !== 'completed') {
```

```
actionItem.querySelector('.complete-action-btn').addEventListener('click',
function() {
        updateActionItemStatus(item.id, 'completed');
      });
    } else {

actionItem.querySelector('.reopen-action-btn').addEventListener('click',
function() {
        updateActionItemStatus(item.id, 'pending');
      });
    }

    return actionItem;
  }

  // View meeting detail
  function viewMeetingDetail(meetingId) {
    const meeting = meetings.find(m => m.id === meetingId);
    if (!meeting) return;

    // Store the selected meeting ID
    selectedMeetingId = meetingId;

    // Populate the details
    document.getElementById('detailMeetingDate').textContent = new
Date(meeting.meetingDate).toLocaleDateString();
    document.getElementById('detailManagerName').textContent =
meeting.managerName;
    document.getElementById('detailNextCheckin').textContent =
meeting.nextCheckInDate ?
      new Date(meeting.nextCheckInDate).toLocaleDateString() : 'Not
scheduled';
```

```
    document.getElementById('detailFeelingOverall').textContent =
meeting.feelingOverall || 'No notes';
    document.getElementById('detailWorkLifeBalance').textContent =
meeting.workLifeBalance || 'No notes';
    document.getElementById('detailAccomplishments').textContent =
meeting.accomplishments || 'No notes';
    document.getElementById('detailFeedbackReflection').textContent =
meeting.feedbackReflection || 'No notes';
    document.getElementById('detailRecentChallenges').textContent =
meeting.recentChallenges || 'No notes';
    document.getElementById('detailBlockers').textContent = meeting.blockers
|| 'No notes';
    document.getElementById('detailGrowthAreas').textContent =
meeting.growthAreas || 'No notes';
    document.getElementById('detailSupportNeeded').textContent =
meeting.supportNeeded || 'No notes';

    // Populate action items
    const actionItemsContainer =
document.getElementById('detailActionItems');
    actionItemsContainer.innerHTML = '';

    if (meeting.actionItems && meeting.actionItems.length > 0) {
      meeting.actionItems.forEach(item => {
        const itemElem = document.createElement('div');
        itemElem.className = 'detail-action-item';

        let statusClass = 'pending';
        if (item.status === 'completed') {
          statusClass = 'completed';
        } else if (item.status === 'overdue') {
          statusClass = 'overdue';
```

```
        }

        itemElem.innerHTML = `
          <p><strong>${item.description}</strong> - Due: ${item.dueDate ? new
Date(item.dueDate).toLocaleDateString() : 'No due date'}</p>
          <p><span class="status-badge
${statusClass}">${item.status}</span></p>
        `;

        actionItemsContainer.appendChild(itemElem);
      });
    } else {
      actionItemsContainer.innerHTML = '<p>No action items</p>';
    }

    // Show the modal
    document.getElementById('meetingDetailModal').style.display = 'flex';
  }

  // Close the meeting detail modal
  function closeDetailModal() {
    document.getElementById('meetingDetailModal').style.display = 'none';
  }

  // Edit meeting (placeholder function)
  function editMeeting(meetingId) {
    // This would be implemented to allow editing existing meetings
    alert('Edit functionality will be implemented in the next release.');
  }

  // Update action item status
  function updateActionItemStatus(itemId, newStatus) {
    google.script.run
```

```
      .withSuccessHandler(function(result) {
        if (result.success) {
          loadActionItems();
        } else {
          alert('Error updating action item: ' + result.message);
        }
      })
      .withFailureHandler(handleError)
      .updateManagerOneOnOneActionItemStatus(itemId, newStatus);
  }

  // Filter action items by status
  function filterActionItems() {
    const status = document.getElementById('filterActionItems').value;
    const items = document.querySelectorAll('#actionItemsList .action-item');

    items.forEach(item => {
      if (status === 'all' || item.getAttribute('data-status') === status) {
        item.style.display = 'flex';
      } else {
        item.style.display = 'none';
      }
    });
  }

  // Search meetings
  function searchMeetings() {
    const searchTerm =
document.getElementById('searchMeetings').value.toLowerCase();
    const items = document.querySelectorAll('#meetingHistory .meeting-item');

    items.forEach(item => {
      const text = item.textContent.toLowerCase();
```

```
      if (text.includes(searchTerm)) {
        item.style.display = 'flex';
      } else {
        item.style.display = 'none';
      }
    });
  }

  // Handle errors
  function handleError(error) {
    console.error('Error:', error);
    alert('An error occurred: ' + (error.message || 'Unknown error'));
  }
 </script>
</body>
</html>
<!DOCTYPE html>
<html>
<head>
 <base target="_top">
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <link
href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500;700&fam
ily=Poppins:wght@500;600;700&display=swap" rel="stylesheet">
 <style>
  /* Variables for Trevor Project color scheme */
  :root {
    /* Primary Colors */
    --trevor-orange: #FF786E;
    --deep-blue: #001A4E;
    --purple: #9A3499;
    --teal: #137F6A;
```

```
    /* Secondary Colors */
    --light-blue: #4F52DE;
    --soft-yellow: #FFAD8D;
    --lavender: #B3AE4A;
    --light-purple: #F54AC;

    /* Tertiary Colors */
    --soft-green: #BAE2CE;
    --pale-yellow: #FFF2DF;
    --light-pink: #FBCBBE;
    --soft-lavender: #D1CFCC;

    /* Gradients */
    --primary-gradient: linear-gradient(135deg, var(--trevor-orange),
var(--light-pink));
    --calm-gradient: linear-gradient(135deg, var(--light-blue),
var(--soft-green));
    --support-gradient: linear-gradient(45deg, var(--purple),
var(--light-purple));
    --background-gradient: linear-gradient(180deg, var(--pale-yellow),
#FFFFFF);

    /* Spacing */
    --spacing-xs: 4px;
    --spacing-sm: 8px;
    --spacing-md: 16px;
    --spacing-lg: 24px;
    --spacing-xl: 32px;

    /* Borders */
    --border-radius-sm: 8px;
    --border-radius-md: 12px;
```

```
  --border-radius-lg: 16px;
}

body {
  font-family: 'Roboto', sans-serif;
  margin: 0;
  padding: 0;
  background-color: var(--pale-yellow);
  color: var(--deep-blue);
}

.container {
  max-width: 800px;
  margin: 0 auto;
  padding: var(--spacing-md);
}

h1, h2, h3 {
  font-family: 'Poppins', sans-serif;
  color: var(--deep-blue);
}

.header {
  margin-bottom: var(--spacing-lg);
  text-align: center;
}

.form-card {
  background: white;
  border-radius: var(--border-radius-md);
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.05);
  padding: var(--spacing-lg);
  margin-bottom: var(--spacing-lg);
```

```
  position: relative;
}

.form-card::before {
  content: '';
  position: absolute;
  left: 0;
  top: 0;
  height: 100%;
  width: 3px;
  background: var(--trevor-orange);
}

.form-group {
  margin-bottom: var(--spacing-md);
}

label {
  display: block;
  font-weight: 500;
  margin-bottom: var(--spacing-xs);
}

.target-info {
  font-size: 0.9em;
  color: #666;
  margin-left: var(--spacing-xs);
  font-style: italic;
}

input, select, textarea {
  width: 100%;
  padding: var(--spacing-sm);
```

```
  border: 1px solid #ddd;
  border-radius: var(--border-radius-sm);
  font-family: 'Roboto', sans-serif;
  font-size: 16px;
}

input[type="number"] {
  width: calc(100% - 16px);
}

textarea {
  resize: vertical;
  min-height: 100px;
}

.btn {
  border: none;
  padding: var(--spacing-md) var(--spacing-lg);
  border-radius: var(--border-radius-sm);
  font-weight: 500;
  cursor: pointer;
  transition: all 0.2s ease;
}

.btn-primary {
  background: var(--primary-gradient);
  color: white;
}

.btn-primary:hover {
  box-shadow: 0 4px 8px rgba(255, 120, 110, 0.3);
  transform: translateY(-2px);
}
```

```css
.btn-secondary {
  background: white;
  color: var(--deep-blue);
  border: 1px solid #ddd;
}

.btn-secondary:hover {
  background: #f5f5f5;
}

.form-actions {
  display: flex;
  justify-content: space-between;
  margin-top: var(--spacing-lg);
}

.success-message {
  background-color: var(--soft-green);
  border-radius: var(--border-radius-sm);
  padding: var(--spacing-md);
  margin-bottom: var(--spacing-md);
  display: none;
}

.error-message {
  background-color: var(--light-pink);
  border-radius: var(--border-radius-sm);
  padding: var(--spacing-md);
  margin-bottom: var(--spacing-md);
  display: none;
}
```

```css
/* Calculated metrics section */
.calculated-metrics {
  margin-top: var(--spacing-md);
  padding-top: var(--spacing-md);
  border-top: 1px dashed #ddd;
}

.metric-row {
  display: flex;
  align-items: center;
  margin-bottom: var(--spacing-sm);
}

.metric-label {
  flex: 0 0 200px;
  font-weight: 500;
}

.metric-value {
  flex: 0 0 100px;
  font-weight: 700;
  color: var(--deep-blue);
}

.metric-target {
  flex: 1;
  color: #666;
  font-style: italic;
  font-size: 0.9em;
}

.status-indicator {
  width: 10px;
```

```
    height: 10px;
    border-radius: 50%;
    margin-right: var(--spacing-sm);
    display: inline-block;
  }

  .status-good {
    background-color: #4CAF50; /* Green */
  }

  .status-warning {
    background-color: #FFC107; /* Amber */
  }

  .status-alert {
    background-color: #F44336; /* Red */
  }
 </style>
</head>
<body>
 <div class="container">
  <div class="header">
    <h1>Enter Daily Metrics</h1>
    <p>Record performance metrics for counselors</p>
  </div>

  <div id="successMessage" class="success-message">
    Metrics saved successfully!
  </div>

  <div id="errorMessage" class="error-message">
    An error occurred while saving metrics. Please try again.
  </div>
```

```
   <div class="form-card">
     <form id="metricsForm">
       <div class="form-group">
         <label for="metricsDate">Date</label>
         <input type="date" id="metricsDate" name="metricsDate" required>
       </div>

       <div class="form-group">
         <label for="callsOffered">Calls Offered</label>
         <input type="number" id="callsOffered" name="callsOffered" min="0"
step="1" required>
       </div>

       <div class="form-group">
         <label for="callsAccepted">Calls Accepted</label>
         <input type="number" id="callsAccepted" name="callsAccepted" min="0"
step="1" required>
       </div>

       <div class="form-group">
         <label for="talkTime">Average Talk Time (minutes)</label>
         <input type="number" id="talkTime" name="talkTime" min="0" step="0.1"
required>
         <span class="target-info">Target: 15-20 minutes</span>
       </div>

       <div class="form-group">
         <label for="afterCallWork">Average After Call Work (minutes)</label>
         <input type="number" id="afterCallWork" name="afterCallWork" min="0"
step="0.1" required>
         <span class="target-info">Target: less than 5 minutes</span>
       </div>
```

```
      <div class="form-group">
        <label for="interactingTime">Interacting Time (hours)</label>
        <input type="number" id="interactingTime" name="interactingTime"
min="0" max="24" step="0.25" required>
        <span class="target-info">Target: 4-6 hours</span>
      </div>

      <div class="form-group">
        <label for="behaviorNotes">Behaviors Impacting Performance</label>
        <textarea id="behaviorNotes" name="behaviorNotes" placeholder="Enter
any notable behaviors or factors that impacted performance
today..."></textarea>
      </div>

      <!-- Calculated metrics section -->
      <div class="calculated-metrics">
        <h3>Calculated Metrics</h3>

        <div class="metric-row">
          <div class="metric-label">Answer Rate:</div>
          <div class="metric-value" id="answerRate">--</div>
          <div class="metric-target">Target: 95%</div>
        </div>

        <div class="metric-row">
          <div class="metric-label">Off Queue Percentage:</div>
          <div class="metric-value" id="offQueuePercent">--</div>
          <div class="metric-target"></div>
        </div>
      </div>

      <div class="form-actions">
```

```
        <button type="button" class="btn btn-secondary"
onclick="google.script.host.close()">Cancel</button>
        <button type="submit" class="btn btn-primary">Save Metrics</button>
      </div>
    </form>
  </div>
 </div>
  <script>
   // Initialize date field with today's date
   document.addEventListener('DOMContentLoaded', function() {
     const today = new Date().toISOString().split('T')[0];
     document.getElementById('metricsDate').value = today;

     // Add form submit handler
     document.getElementById('metricsForm').addEventListener('submit',
function(e) {
       e.preventDefault();
       saveMetrics();
     });

     // Add event listeners for real-time calculation
     document.getElementById('callsOffered').addEventListener('input',
calculateMetrics);
     document.getElementById('callsAccepted').addEventListener('input',
calculateMetrics);
     document.getElementById('onQueue').addEventListener('input',
calculateMetrics);
   });

   // Calculate derived metrics
   function calculateMetrics() {
     const callsOffered =
parseFloat(document.getElementById('callsOffered').value) || 0;
```

```
    const callsAccepted =
parseFloat(document.getElementById('callsAccepted').value) || 0;
    const onQueue = parseFloat(document.getElementById('onQueue').value) ||
0;

    // Calculate answer rate
    let answerRate = 0;
    if (callsOffered > 0) {
      answerRate = (callsAccepted / callsOffered * 100).toFixed(1);
    }
    document.getElementById('answerRate').textContent = answerRate + '%';

    // Calculate off queue percentage
    const offQueuePercent = (100 - onQueue).toFixed(1);
    document.getElementById('offQueuePercent').textContent = offQueuePercent
+ '%';

    // Add visual indicators based on targets
    if (answerRate >= 95) {
      document.getElementById('answerRate').innerHTML = '<span
class="status-indicator status-good"></span>' + answerRate + '%';
    } else if (answerRate >= 90) {
      document.getElementById('answerRate').innerHTML = '<span
class="status-indicator status-warning"></span>' + answerRate + '%';
    } else if (answerRate > 0) {
      document.getElementById('answerRate').innerHTML = '<span
class="status-indicator status-alert"></span>' + answerRate + '%';
    }
  }

  // Save metrics to the spreadsheet
  function saveMetrics() {
    // Get form data
```

```
    const formData = {
      date: document.getElementById('metricsDate').value,
      callsOffered: parseFloat(document.getElementById('callsOffered').value)
|| 0,
      callsAccepted:
parseFloat(document.getElementById('callsAccepted').value) || 0,
      talkTime: parseFloat(document.getElementById('talkTime').value) || 0,
      afterCallWork:
parseFloat(document.getElementById('afterCallWork').value) || 0,
      onQueue: parseFloat(document.getElementById('onQueue').value) || 0,
      interactingTime:
parseFloat(document.getElementById('interactingTime').value) || 0,
      behaviorNotes: document.getElementById('behaviorNotes').value
    };

    // Add derived metrics
    formData.answerRate = formData.callsOffered > 0 ?
      (formData.callsAccepted / formData.callsOffered * 100).toFixed(1) : 0;
    formData.offQueuePercent = (100 - formData.onQueue).toFixed(1);

    // Show loading state
    const submitButton = document.querySelector('button[type="submit"]');
    const originalText = submitButton.textContent;
    submitButton.textContent = 'Saving...';
    submitButton.disabled = true;

    // Submit to the server
    google.script.run
      .withSuccessHandler(function(result) {
        // Reset button state
        submitButton.textContent = originalText;
        submitButton.disabled = false;
```

```
          if (result.success) {
            // Show success message
            document.getElementById('successMessage').style.display = 'block';
            document.getElementById('errorMessage').style.display = 'none';

            // Reset form
            document.getElementById('metricsForm').reset();

            // Set today's date again
            document.getElementById('metricsDate').value = new
Date().toISOString().split('T')[0];

            // Reset calculated fields
            document.getElementById('answerRate').textContent = '--';
            document.getElementById('offQueuePercent').textContent = '--';
          } else {
            // Show error message
            document.getElementById('errorMessage').style.display = 'block';
            document.getElementById('successMessage').style.display = 'none';
            document.getElementById('errorMessage').textContent =
result.message || 'An error occurred while saving metrics. Please try again.';
          }
        })
        .withFailureHandler(function(error) {
          // Reset button state
          submitButton.textContent = originalText;
          submitButton.disabled = false;

          // Show error message
          document.getElementById('errorMessage').style.display = 'block';
          document.getElementById('successMessage').style.display = 'none';
          document.getElementById('errorMessage').textContent = error.message
|| 'An error occurred while saving metrics. Please try again.';
```

```
      })
      .saveMetrics(formData);
  }
 </script>
</body>
</html>
<!DOCTYPE html>
<html>
<head>
 <base target="_top">
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <link
href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500;700&fam
ily=Poppins:wght@500;600;700&display=swap" rel="stylesheet">
 <style>
   /* Variables for 988 Lifeline LGBTQIA+ Services color scheme */
   :root {
     /* Primary Colors */
     --trevor-orange: #FF786E;
     --deep-blue: #001A4E;
     --purple: #9A3499;
     --teal: #137F6A;

     /* Secondary Colors */
     --light-blue: #4F52DE;
     --soft-yellow: #FFAD8D;
     --lavender: #B3AE4A;
     --light-purple: #F54AC;

     /* Tertiary Colors */
     --soft-green: #BAE2CE;
     --pale-yellow: #FFF2DF;
```

```
    --light-pink: #FBCBBE;
    --soft-lavender: #D1CFCC;

    /* Gradients */
    --primary-gradient: linear-gradient(135deg, var(--trevor-orange),
var(--light-pink));
    --calm-gradient: linear-gradient(135deg, var(--light-blue),
var(--soft-green));
    --support-gradient: linear-gradient(45deg, var(--purple),
var(--light-purple));
    --background-gradient: linear-gradient(180deg, var(--pale-yellow),
#FFFFFF);

    /* Spacing */
    --spacing-xs: 4px;
    --spacing-sm: 8px;
    --spacing-md: 16px;
    --spacing-lg: 24px;
    --spacing-xl: 32px;

    /* Borders */
    --border-radius-sm: 8px;
    --border-radius-md: 12px;
    --border-radius-lg: 16px;
  }

  body {
    font-family: 'Roboto', sans-serif;
    margin: 0;
    padding: 0;
    background-color: var(--pale-yellow);
    color: var(--deep-blue);
  }
```

```css
.container {
  max-width: 1200px;
  margin: 0 auto;
  padding: var(--spacing-md);
}

.header {
  margin-bottom: var(--spacing-lg);
  text-align: center;
}

h1, h2, h3, h4 {
  font-family: 'Poppins', sans-serif;
  color: var(--deep-blue);
}

.report-card {
  background: white;
  border-radius: var(--border-radius-md);
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.05);
  padding: var(--spacing-lg);
  position: relative;
  margin-bottom: var(--spacing-lg);
}

.report-card::before {
  content: '';
  position: absolute;
  left: 0;
  top: 0;
  height: 100%;
  width: 3px;
```

```
  background: var(--light-blue);
}

.filters {
  display: flex;
  flex-wrap: wrap;
  gap: var(--spacing-md);
  margin-bottom: var(--spacing-lg);
  padding: var(--spacing-md);
  background-color: white;
  border-radius: var(--border-radius-md);
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.05);
}

.filter-group {
  flex: 1;
  min-width: 200px;
}

.filter-group label {
  display: block;
  font-weight: 500;
  margin-bottom: var(--spacing-xs);
}

.filter-group select, .filter-group input {
  width: 100%;
  padding: var(--spacing-sm);
  border: 1px solid #ddd;
  border-radius: var(--border-radius-sm);
  font-family: 'Roboto', sans-serif;
}
```

```css
.btn {
  border: none;
  padding: var(--spacing-md) var(--spacing-lg);
  border-radius: var(--border-radius-sm);
  font-weight: 500;
  cursor: pointer;
  transition: all 0.2s ease;
}

.btn-primary {
  background: var(--primary-gradient);
  color: white;
}

.btn-primary:hover {
  box-shadow: 0 4px 8px rgba(255, 120, 110, 0.3);
  transform: translateY(-2px);
}

.btn-secondary {
  background: white;
  color: var(--deep-blue);
  border: 1px solid #ddd;
}

.btn-secondary:hover {
  background: #f5f5f5;
}

.chart-container {
  height: 400px;
  margin-bottom: var(--spacing-lg);
  position: relative;
```

```css
}

.chart-placeholder {
  display: flex;
  align-items: center;
  justify-content: center;
  height: 100%;
  background-color: #f5f5f5;
  border-radius: var(--border-radius-md);
  color: #777;
  font-size: 16px;
}

.metrics-grid {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(220px, 1fr));
  gap: var(--spacing-md);
  margin-bottom: var(--spacing-lg);
}

.metric-card {
  background: white;
  border-radius: var(--border-radius-sm);
  padding: var(--spacing-md);
  text-align: center;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.05);
  transition: transform 0.3s ease;
}

.metric-card:hover {
  transform: translateY(-3px);
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
}
```

```css
.metric-value {
  font-size: 32px;
  font-weight: 700;
  margin: var(--spacing-sm) 0;
  color: var(--trevor-orange);
}

.metric-label {
  font-size: 14px;
  color: #666;
}

.metric-target {
  font-size: 12px;
  color: var(--teal);
  font-weight: 500;
}

.table-container {
  margin-top: var(--spacing-lg);
  overflow-x: auto;
}

table {
  width: 100%;
  border-collapse: collapse;
  margin-bottom: var(--spacing-lg);
}

thead {
  background-color: var(--deep-blue);
  color: white;
```

```css
}

th, td {
  padding: var(--spacing-sm) var(--spacing-md);
  text-align: left;
  border: 1px solid #ddd;
}

tr:nth-child(even) {
  background-color: #f9f9f9;
}

tr:hover {
  background-color: #f1f1f1;
}

.tab-container {
  margin-bottom: var(--spacing-lg);
}

.tabs {
  display: flex;
  border-bottom: 1px solid #ddd;
  margin-bottom: var(--spacing-md);
}

.tab {
  padding: var(--spacing-md) var(--spacing-lg);
  cursor: pointer;
  border: 1px solid transparent;
  border-bottom: none;
  border-radius: var(--border-radius-sm) var(--border-radius-sm) 0 0;
  margin-right: var(--spacing-xs);
```

```
  background-color: transparent;
  font-weight: 500;
  transition: all 0.2s ease;
}

.tab.active {
  background-color: white;
  border-color: #ddd;
  color: var(--trevor-orange);
}

.tab-content {
  display: none;
}

.tab-content.active {
  display: block;
}

.actions {
  display: flex;
  justify-content: space-between;
  margin-top: var(--spacing-lg);
}

.counselor-filter {
  display: flex;
  align-items: center;
  flex-wrap: wrap;
  gap: var(--spacing-sm);
  margin-bottom: var(--spacing-md);
}
```

```css
.counselor-checkbox {
  display: flex;
  align-items: center;
  background-color: #f5f5f5;
  padding: var(--spacing-xs) var(--spacing-sm);
  border-radius: var(--border-radius-sm);
}

.counselor-checkbox input {
  margin-right: var(--spacing-xs);
}

.loading {
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background-color: rgba(255, 255, 255, 0.8);
  display: flex;
  align-items: center;
  justify-content: center;
  z-index: 100;
  border-radius: var(--border-radius-md);
}

.loading-spinner {
  border: 4px solid #f3f3f3;
  border-top: 4px solid var(--trevor-orange);
  border-radius: 50%;
  width: 30px;
  height: 30px;
  animation: spin 1s linear infinite;
```

```
}

@keyframes spin {
  0% { transform: rotate(0deg); }
  100% { transform: rotate(360deg); }
}

@media (max-width: 768px) {
  .metrics-grid {
    grid-template-columns: 1fr 1fr;
  }

  .filters {
    flex-direction: column;
  }

  .actions {
    flex-direction: column;
    gap: var(--spacing-md);
  }

  .tab {
    padding: var(--spacing-sm) var(--spacing-md);
  }
}

@media (max-width: 480px) {
  .metrics-grid {
    grid-template-columns: 1fr;
  }

  .tabs {
    flex-wrap: wrap;
```

```
    }
  }
 </style>
</head>
<body>
 <div class="container">
   <div class="header">
     <h1>Call Metrics Reports</h1>
     <p>Performance analytics for the 988 Lifeline LGBTQIA+ team</p>
   </div>

   <div class="filters">
     <div class="filter-group">
       <label for="dateRange">Date Range</label>
       <select id="dateRange" onchange="updateDateFields()">
         <option value="last7days">Last 7 Days</option>
         <option value="last30days" selected>Last 30 Days</option>
         <option value="last90days">Last 90 Days</option>
         <option value="thisMonth">This Month</option>
         <option value="lastMonth">Last Month</option>
         <option value="custom">Custom Range</option>
       </select>
     </div>

     <div class="filter-group date-field" id="startDateGroup">
       <label for="startDate">Start Date</label>
       <input type="date" id="startDate">
     </div>

     <div class="filter-group date-field" id="endDateGroup">
       <label for="endDate">End Date</label>
       <input type="date" id="endDate">
     </div>
```

```
    <div class="filter-group">
      <label for="team">Team</label>
      <select id="team">
        <option value="all" selected>All Teams</option>
        <option value="digital">Digital Team</option>
        <option value="lifeline">Lifeline Team</option>
      </select>
    </div>

    <div class="filter-group">
      <label> </label>
      <button class="btn btn-primary" onclick="generateReport()">Generate
Report</button>
    </div>
  </div>

  <div class="tab-container">
    <div class="tabs">
      <button class="tab active"
onclick="switchTab('overview')">Overview</button>
      <button class="tab" onclick="switchTab('counselors')">Counselor
Performance</button>
      <button class="tab" onclick="switchTab('trends')">Trends
Analysis</button>
      <button class="tab" onclick="switchTab('detailed')">Detailed
Data</button>
    </div>

    <!-- Overview Tab -->
    <div id="overview" class="tab-content active">
      <div class="report-card">
        <h2>Key Performance Metrics</h2>
```

```
        <p>Summary for <span id="reportPeriod">the last 30 days</span></p>

        <div class="metrics-grid">
          <div class="metric-card">
            <div class="metric-label">Answer Rate</div>
            <div class="metric-value" id="answerRate">95.2%</div>
            <div class="metric-target">Target: 95%</div>
          </div>

          <div class="metric-card">
            <div class="metric-label">Average Talk Time</div>
            <div class="metric-value" id="avgTalkTime">18:45</div>
            <div class="metric-target">Target: 15-25 min</div>
          </div>

          <div class="metric-card">
            <div class="metric-label">On-Queue Percentage</div>
            <div class="metric-value" id="onQueuePercent">89.3%</div>
            <div class="metric-target">Target: 85%</div>
          </div>

          <div class="metric-card">
            <div class="metric-label">Total Calls</div>
            <div class="metric-value" id="totalCalls">1,287</div>
            <div class="metric-target">Volume Trend: +5.2%</div>
          </div>

          <div class="metric-card">
            <div class="metric-label">After-Call Work</div>
            <div class="metric-value" id="afterCallWork">3:12</div>
            <div class="metric-target">Target: < 5 min</div>
          </div>
```

```
        <div class="metric-card">
          <div class="metric-label">Quality Score</div>
          <div class="metric-value" id="qualityScore">74.6</div>
          <div class="metric-target">Target: 70+</div>
        </div>
      </div>

      <div class="chart-container">
        <div id="trendChart" class="chart-placeholder">
          <div>Loading call volume trend chart...</div>
        </div>
      </div>
    </div>
  </div>

  <!-- Counselor Performance Tab -->
  <div id="counselors" class="tab-content">
    <div class="report-card">
      <h2>Counselor Performance Metrics</h2>

      <div class="counselor-filter">
        <span><strong>Filter Counselors:</strong></span>
        <div class="counselor-checkbox">
          <input type="checkbox" id="activeOnly" checked>
          <label for="activeOnly">Active Only</label>
        </div>
        <div class="counselor-checkbox">
          <input type="checkbox" id="belowTarget">
          <label for="belowTarget">Below Target</label>
        </div>
        <div class="counselor-checkbox">
          <input type="checkbox" id="digitalTeam" checked>
          <label for="digitalTeam">Digital Team</label>
```

```
        </div>
        <div class="counselor-checkbox">
          <input type="checkbox" id="lifelineTeam" checked>
          <label for="lifelineTeam">Lifeline Team</label>
        </div>
      </div>

      <div class="chart-container">
        <div id="counselorChart" class="chart-placeholder">
          <div>Loading counselor performance comparison chart...</div>
        </div>
      </div>

      <div class="table-container">
        <table id="counselorTable">
          <thead>
            <tr>
              <th>Counselor</th>
              <th>Team</th>
              <th>Answer Rate</th>
              <th>Avg Talk Time</th>
              <th>On-Queue %</th>
              <th>Calls/Shift</th>
              <th>Quality Score</th>
            </tr>
          </thead>
          <tbody>
            <tr>
              <td>Loading counselor data...</td>
              <td></td>
              <td></td>
              <td></td>
              <td></td>
```

```
              <td></td>
              <td></td>
            </tr>
          </tbody>
        </table>
      </div>
    </div>
  </div>

  <!-- Trends Analysis Tab -->
  <div id="trends" class="tab-content">
    <div class="report-card">
      <h2>Trends Analysis</h2>

      <div class="chart-container">
        <div id="weekdayChart" class="chart-placeholder">
          <div>Loading weekday analysis chart...</div>
        </div>
      </div>

      <div class="chart-container">
        <div id="hourlyChart" class="chart-placeholder">
          <div>Loading hourly distribution chart...</div>
        </div>
      </div>

      <div class="metrics-grid">
        <div class="metric-card">
          <div class="metric-label">Busiest Day</div>
          <div class="metric-value" id="busiestDay">Monday</div>
          <div class="metric-target">+32% vs. average</div>
        </div>
```

```
        <div class="metric-card">
          <div class="metric-label">Busiest Hour</div>
          <div class="metric-value" id="busiestHour">7pm</div>
          <div class="metric-target">+45% vs. average</div>
        </div>

        <div class="metric-card">
          <div class="metric-label">Slowest Day</div>
          <div class="metric-value" id="slowestDay">Saturday</div>
          <div class="metric-target">-28% vs. average</div>
        </div>

        <div class="metric-card">
          <div class="metric-label">Slowest Hour</div>
          <div class="metric-value" id="slowestHour">4am</div>
          <div class="metric-target">-65% vs. average</div>
        </div>
      </div>
    </div>
  </div>

  <!-- Detailed Data Tab -->
  <div id="detailed" class="tab-content">
    <div class="report-card">
      <h2>Detailed Metrics Data</h2>

      <div class="table-container">
        <table id="detailedTable">
          <thead>
            <tr>
              <th>Date</th>
              <th>Calls Offered</th>
              <th>Calls Accepted</th>
```

```
                <th>Answer Rate</th>
                <th>Talk Time (min)</th>
                <th>After Call (min)</th>
                <th>On Queue %</th>
              </tr>
            </thead>
            <tbody>
              <tr>
                <td>Loading detailed data...</td>
                <td></td>
                <td></td>
                <td></td>
                <td></td>
                <td></td>
                <td></td>
              </tr>
            </tbody>
          </table>
        </div>
      </div>
    </div>
  </div>

  <div class="actions">
    <button class="btn btn-secondary"
onclick="google.script.host.close()">Close</button>
    <button class="btn btn-primary" onclick="exportToSpreadsheet()">Export to
Spreadsheet</button>
  </div>
 </div>
  <script>
  // Initialize on page load
  document.addEventListener('DOMContentLoaded', function() {
```

```
  // Set default date range
  updateDateFields();

  // Generate initial report
  generateReport();
});

// Update date fields based on selected range
function updateDateFields() {
  const dateRange = document.getElementById('dateRange').value;
  const startDateGroup = document.getElementById('startDateGroup');
  const endDateGroup = document.getElementById('endDateGroup');
  const startDate = document.getElementById('startDate');
  const endDate = document.getElementById('endDate');

  // Hide date fields by default
  startDateGroup.style.display = 'none';
  endDateGroup.style.display = 'none';

  // Calculate dates based on selection
  const today = new Date();
  let start = new Date();
  let end = new Date();

  switch (dateRange) {
    case 'last7days':
      start.setDate(today.getDate() - 7);
      break;
    case 'last30days':
      start.setDate(today.getDate() - 30);
      break;
    case 'last90days':
      start.setDate(today.getDate() - 90);
```

```
        break;
      case 'thisMonth':
        start = new Date(today.getFullYear(), today.getMonth(), 1);
        end = new Date(today.getFullYear(), today.getMonth() + 1, 0);
        break;
      case 'lastMonth':
        start = new Date(today.getFullYear(), today.getMonth() - 1, 1);
        end = new Date(today.getFullYear(), today.getMonth(), 0);
        break;
      case 'custom':
        startDateGroup.style.display = 'block';
        endDateGroup.style.display = 'block';
        return;
    }

    // Format dates for input fields
    startDate.value = formatDate(start);
    endDate.value = formatDate(end);
}

// Format date as YYYY-MM-DD
function formatDate(date) {
    const year = date.getFullYear();
    const month = String(date.getMonth() + 1).padStart(2, '0');
    const day = String(date.getDate()).padStart(2, '0');
    return `${year}-${month}-${day}`;
}

// Switch between tabs
function switchTab(tabId) {
    // Hide all tab contents
    document.querySelectorAll('.tab-content').forEach(tab => {
        tab.classList.remove('active');
```

```
    });

    // Deactivate all tabs
    document.querySelectorAll('.tab').forEach(tab => {
      tab.classList.remove('active');
    });

    // Activate selected tab
    document.getElementById(tabId).classList.add('active');

document.querySelector(`.tab[onclick="switchTab('${tabId}')"]`).classList.add(
'active');
  }

  // Generate report based on selected filters
  function generateReport() {
    // Show loading indicators
    document.querySelectorAll('.chart-placeholder').forEach(chart => {
      chart.innerHTML = '<div class="loading"><div
class="loading-spinner"></div></div>';
    });

    // Get filter values
    const startDate = document.getElementById('startDate').value;
    const endDate = document.getElementById('endDate').value;
    const team = document.getElementById('team').value;
    const dateRange = document.getElementById('dateRange').value;

    // Update report period text
    updateReportPeriodText();

    // Fetch metrics data from server
    google.script.run
```

```
      .withSuccessHandler(handleReportData)
      .withFailureHandler(handleError)
      .getMetricsReport({
        startDate: startDate,
        endDate: endDate,
        team: team
      });
  }

  // Update the report period text based on selected date range
  function updateReportPeriodText() {
    const dateRange = document.getElementById('dateRange').value;
    const startDate = document.getElementById('startDate').value;
    const endDate = document.getElementById('endDate').value;
    let periodText = '';

    switch (dateRange) {
      case 'last7days':
        periodText = 'the last 7 days';
        break;
      case 'last30days':
        periodText = 'the last 30 days';
        break;
      case 'last90days':
        periodText = 'the last 90 days';
        break;
      case 'thisMonth':
        periodText = 'this month';
        break;
      case 'lastMonth':
        periodText = 'last month';
        break;
      case 'custom':
```

```
        periodText = `${formatDateDisplay(startDate)} to
${formatDateDisplay(endDate)}`;
        break;
    }

    document.getElementById('reportPeriod').textContent = periodText;
  }

  // Format date for display (e.g., Jan 15, 2025)
  function formatDateDisplay(dateString) {
    const date = new Date(dateString);
    const months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug',
'Sep', 'Oct', 'Nov', 'Dec'];
    return `${months[date.getMonth()]} ${date.getDate()},
${date.getFullYear()}`;
  }

  // Handle report data from server
  function handleReportData(data) {
    // If no data, show empty message
    if (!data || !data.summary) {
      handleError('No data found for the selected period');
      return;
    }

    // Update summary metrics
    document.getElementById('answerRate').textContent =
(data.summary.answerRate || 0).toFixed(1) + '%';
    document.getElementById('avgTalkTime').textContent =
formatTime(data.summary.avgTalkTime || 0);
    document.getElementById('onQueuePercent').textContent =
(data.summary.onQueuePercent || 0).toFixed(1) + '%';
```

```
    document.getElementById('totalCalls').textContent =
formatNumber(data.summary.totalCalls || 0);
    document.getElementById('afterCallWork').textContent =
formatTime(data.summary.afterCallWork || 0);
    document.getElementById('qualityScore').textContent =
(data.summary.qualityScore || 0).toFixed(1);

    // Update trends metrics if available
    if (data.trends) {
      document.getElementById('busiestDay').textContent =
data.trends.busiestDay || 'N/A';
      document.getElementById('busiestHour').textContent =
data.trends.busiestHour || 'N/A';
      document.getElementById('slowestDay').textContent =
data.trends.slowestDay || 'N/A';
      document.getElementById('slowestHour').textContent =
data.trends.slowestHour || 'N/A';
    }

    // Update counselor table if available
    if (data.counselors && data.counselors.length > 0) {
      updateCounselorTable(data.counselors);
    }

    // Update detailed data table if available
    if (data.detailed && data.detailed.length > 0) {
      updateDetailedTable(data.detailed);
    }

    // Simulate chart data (would be replaced with actual chart rendering)
    simulateCharts();
  }
```

```
  // Update counselor table with data
  function updateCounselorTable(counselors) {
    const table = document.getElementById('counselorTable');
    const tbody = table.querySelector('tbody');
    tbody.innerHTML = '';

    counselors.forEach(counselor => {
      const row = document.createElement('tr');

      // Add class for below target values
      const answerRateClass = counselor.answerRate < 95 ? 'below-target' :
'';
      const qualityScoreClass = counselor.qualityScore < 70 ? 'below-target'
: '';
      const onQueueClass = counselor.onQueuePercent < 85 ? 'below-target' :
'';

      row.innerHTML = `
        <td>${counselor.name}</td>
        <td>${counselor.team}</td>
        <td
class="${answerRateClass}">${counselor.answerRate.toFixed(1)}%</td>
        <td>${formatTime(counselor.avgTalkTime)}</td>
        <td
class="${onQueueClass}">${counselor.onQueuePercent.toFixed(1)}%</td>
        <td>${counselor.callsPerShift.toFixed(1)}</td>
        <td
class="${qualityScoreClass}">${counselor.qualityScore.toFixed(1)}</td>
      `;

      tbody.appendChild(row);
    });
  }
```

```
// Update detailed data table
function updateDetailedTable(detailedData) {
  const table = document.getElementById('detailedTable');
  const tbody = table.querySelector('tbody');
  tbody.innerHTML = '';

  detailedData.forEach(day => {
    const row = document.createElement('tr');

    // Calculate answer rate
    const answerRate = day.callsAccepted / day.callsOffered * 100;

    row.innerHTML = `
      <td>${formatDateDisplay(day.date)}</td>
      <td>${formatNumber(day.callsOffered)}</td>
      <td>${formatNumber(day.callsAccepted)}</td>
      <td>${answerRate.toFixed(1)}%</td>
      <td>${formatTime(day.talkTime)}</td>
      <td>${formatTime(day.afterCallWork)}</td>
      <td>${day.onQueuePercent.toFixed(1)}%</td>
    `;

    tbody.appendChild(row);
  });
}

// Format number with commas
function formatNumber(num) {
  return num.toString().replace(/\B(?=(\d{3})+(?!\d))/g, ",");
}

// Format minutes as MM:SS
```

```
    function formatTime(minutes) {
      const mins = Math.floor(minutes);
      const secs = Math.round((minutes - mins) * 60);
      return `${mins}:${secs.toString().padStart(2, '0')}`;
    }

    // Handle errors
    function handleError(error) {
      console.error('Error:', error);

      // Clear loading indicators
      document.querySelectorAll('.chart-placeholder').forEach(chart => {
        chart.innerHTML = `<div>Error loading data: ${error}</div>`;
      });

      // Show error message for tables

document.getElementById('counselorTable').querySelector('tbody').innerHTML = `
        <tr><td colspan="7">Error loading data: ${error}</td></tr>
      `;

      document.getElementById('detailedTable').querySelector('tbody').innerHTML
= `
        <tr><td colspan="7">Error loading data: ${error}</td></tr>
      `;
    }

    // Simulate chart visualizations
    function simulateCharts() {
      // In a real implementation, these would use actual charting libraries
      // like Chart.js or Google Charts
      document.getElementById('trendChart').innerHTML = `
        <div style="text-align: center;">
```

```
        <img
src="https://via.placeholder.com/800x300?text=Call+Volume+Trend+Chart"
alt="Call Volume Trend" style="max-width: 100%; height: auto;">
      </div>
    `;

    document.getElementById('counselorChart').innerHTML = `
      <div style="text-align: center;">
        <img
src="https://via.placeholder.com/800x300?text=Counselor+Performance+Chart"
alt="Counselor Performance" style="max-width: 100%; height: auto;">
      </div>
    `;

    document.getElementById('weekdayChart').innerHTML = `
      <div style="text-align: center;">
        <img
src="https://via.placeholder.com/800x300?text=Weekday+Distribution+Chart"
alt="Weekday Distribution" style="max-width: 100%; height: auto;">
      </div>
    `;

    document.getElementById('hourlyChart').innerHTML = `
      <div style="text-align: center;">
        <img
src="https://via.placeholder.com/800x300?text=Hourly+Distribution+Chart"
alt="Hourly Distribution" style="max-width: 100%; height: auto;">
      </div>
    `;
  }

  // Export report to spreadsheet
  function exportToSpreadsheet() {
```

```
    // Get filter values
    const startDate = document.getElementById('startDate').value;
    const endDate = document.getElementById('endDate').value;
    const team = document.getElementById('team').value;

    // Show loading message
    alert('Exporting report to spreadsheet...');

    // Call server function to create spreadsheet
    google.script.run
      .withSuccessHandler(function(result) {
        alert('Report exported successfully!');
      })
      .withFailureHandler(function(error) {
        alert('Error exporting report: ' + error);
      })
      .exportMetricsReport({
        startDate: startDate,
        endDate: endDate,
        team: team
      });
  }
 </script>
</body>
</html>
<!DOCTYPE html>
<html>
<head>
 <base target="_top">
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
 <link
href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500;700&fam
ily=Poppins:wght@500;600;700&display=swap" rel="stylesheet">
 <style>
   /* Variables for Trevor Project color scheme */
   :root {
     /* Primary Colors */
     --trevor-orange: #FF786E;
     --deep-blue: #001A4E;
     --purple: #9A3499;
     --teal: #137F6A;

     /* Secondary Colors */
     --light-blue: #4F52DE;
     --soft-yellow: #FFAD8D;
     --lavender: #B3AE4A;
     --light-purple: #F54AC;

     /* Tertiary Colors */
     --soft-green: #BAE2CE;
     --pale-yellow: #FFF2DF;
     --light-pink: #FBCBBE;
     --soft-lavender: #D1CFCC;

     /* Gradients */
     --primary-gradient: linear-gradient(135deg, var(--trevor-orange),
var(--light-pink));
     --calm-gradient: linear-gradient(135deg, var(--light-blue),
var(--soft-green));
     --support-gradient: linear-gradient(45deg, var(--purple),
var(--light-purple));
     --background-gradient: linear-gradient(180deg, var(--pale-yellow),
#FFFFFF);
```

```
  /* Spacing */
  --spacing-xs: 4px;
  --spacing-sm: 8px;
  --spacing-md: 16px;
  --spacing-lg: 24px;
  --spacing-xl: 32px;

  /* Borders */
  --border-radius-sm: 8px;
  --border-radius-md: 12px;
  --border-radius-lg: 16px;
}

body {
  font-family: 'Roboto', sans-serif;
  margin: 0;
  padding: 0;
  background-color: var(--pale-yellow);
  color: var(--deep-blue);
}

.container {
  max-width: 900px;
  margin: 0 auto;
  padding: var(--spacing-md);
}

h1, h2, h3, h4 {
  font-family: 'Poppins', sans-serif;
  color: var(--deep-blue);
  margin-top: 0;
}
```

```css
.header {
  margin-bottom: var(--spacing-lg);
  text-align: center;
}

.card {
  background: white;
  border-radius: var(--border-radius-md);
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.05);
  padding: var(--spacing-lg);
  margin-bottom: var(--spacing-lg);
  position: relative;
}

.card::before {
  content: '';
  position: absolute;
  left: 0;
  top: 0;
  height: 100%;
  width: 3px;
  background: var(--trevor-orange);
}

.form-group {
  margin-bottom: var(--spacing-md);
}

label {
  display: block;
  font-weight: 500;
  margin-bottom: var(--spacing-xs);
```

```css
}

input, select, textarea {
  width: 100%;
  padding: var(--spacing-sm);
  border: 1px solid #ddd;
  border-radius: var(--border-radius-sm);
  font-family: 'Roboto', sans-serif;
  font-size: 16px;
}

textarea {
  resize: vertical;
  min-height: 80px;
}

.btn {
  border: none;
  padding: var(--spacing-md) var(--spacing-lg);
  border-radius: var(--border-radius-sm);
  font-weight: 500;
  cursor: pointer;
  transition: all 0.2s ease;
}

.btn-primary {
  background: var(--primary-gradient);
  color: white;
}

.btn-primary:hover {
  box-shadow: 0 4px 8px rgba(255, 120, 110, 0.3);
  transform: translateY(-2px);
```

```css
}

.btn-secondary {
  background: white;
  color: var(--deep-blue);
  border: 1px solid #ddd;
}

.btn-secondary:hover {
  background: #f5f5f5;
}

.btn-sm {
  font-size: 14px;
  padding: var(--spacing-xs) var(--spacing-sm);
}

.center {
  text-align: center;
}

.section-actions {
  display: flex;
  justify-content: space-between;
  margin-top: var(--spacing-md);
}

.form-actions {
  display: flex;
  justify-content: space-between;
  margin-top: var(--spacing-lg);
}
```

```css
.tab-container {
  margin-bottom: var(--spacing-lg);
}

.tabs {
  display: flex;
  margin-bottom: var(--spacing-md);
  border-bottom: 1px solid #ddd;
}

.tab {
  padding: var(--spacing-sm) var(--spacing-md);
  cursor: pointer;
  border-bottom: 3px solid transparent;
  transition: all 0.2s ease;
}

.tab.active {
  border-bottom-color: var(--trevor-orange);
  font-weight: 500;
}

.tab-content {
  display: none;
}

.tab-content.active {
  display: block;
}

.success-message {
  background-color: var(--soft-green);
  border-radius: var(--border-radius-sm);
```

```
  padding: var(--spacing-md);
  margin-bottom: var(--spacing-md);
  display: none;
}

.error-message {
  background-color: var(--light-pink);
  border-radius: var(--border-radius-sm);
  padding: var(--spacing-md);
  margin-bottom: var(--spacing-md);
  display: none;
}

.counselor-selector {
  padding: var(--spacing-md);
  margin-bottom: var(--spacing-lg);
  background: white;
  border-radius: var(--border-radius-md);
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.05);
}

.rating-group {
  display: flex;
  align-items: center;
  margin-bottom: var(--spacing-md);
}

.rating-label {
  flex: 0 0 200px;
  font-weight: 500;
}

.rating-options {
```

```
  flex: 1;
  display: flex;
  gap: var(--spacing-md);
}

.rating-options label {
  display: flex;
  align-items: center;
  font-weight: normal;
  margin: 0;
  cursor: pointer;
}

.rating-options input {
  width: auto;
  margin-right: var(--spacing-xs);
}

.section-header {
  margin-bottom: var(--spacing-md);
  padding-bottom: var(--spacing-xs);
  border-bottom: 1px solid #eee;
}

.previous-meetings {
  margin-top: var(--spacing-lg);
}

.meeting-item {
  padding: var(--spacing-sm) var(--spacing-md);
  margin-bottom: var(--spacing-sm);
  border-bottom: 1px solid #eee;
  cursor: pointer;
```

```
  transition: background-color 0.2s ease;
}

.meeting-item:hover {
  background: #f9f9f9;
}

.meeting-date {
  font-weight: 500;
  margin-bottom: 4px;
}

.meeting-summary {
  font-size: 0.9em;
  color: #666;
  margin-top: var(--spacing-xs);
}

.meeting-next-checkin {
  font-size: 0.85em;
  color: var(--trevor-orange);
  margin-top: 4px;
}

/* Counselor details card */
.counselor-details-card {
  background: white;
  border-radius: var(--border-radius-md);
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.05);
  padding: var(--spacing-md);
  margin-bottom: var(--spacing-lg);
  display: flex;
  flex-direction: column;
```

```
}

.counselor-details-header {
  display: flex;
  align-items: center;
  margin-bottom: var(--spacing-md);
}

.counselor-avatar {
  width: 60px;
  height: 60px;
  border-radius: 50%;
  background: var(--primary-gradient);
  color: white;
  display: flex;
  align-items: center;
  justify-content: center;
  font-size: 24px;
  font-weight: bold;
  margin-right: var(--spacing-md);
}

.counselor-info {
  flex: 1;
}

.counselor-name {
  font-size: 20px;
  font-weight: 600;
  margin-bottom: 4px;
}

.counselor-status {
```

```
  display: inline-block;
  padding: 4px 8px;
  border-radius: 12px;
  font-size: 12px;
  font-weight: 500;
  text-transform: uppercase;
}

.status-active {
  background-color: var(--soft-green);
  color: var(--deep-blue);
}

.status-inactive {
  background-color: var(--light-purple);
  color: white;
}

.status-leave {
  background-color: var(--soft-yellow);
  color: var(--deep-blue);
}

.counselor-metadata {
  display: flex;
  flex-wrap: wrap;
  gap: var(--spacing-md);
}

.metadata-item {
  flex: 1;
  min-width: 150px;
}
```

```css
.metadata-label {
  font-size: 12px;
  color: #666;
  margin-bottom: 2px;
}

.metadata-value {
  font-weight: 500;
}

.loading-spinner {
  display: inline-block;
  width: 20px;
  height: 20px;
  border: 3px solid rgba(255, 120, 110, 0.3);
  border-radius: 50%;
  border-top-color: var(--trevor-orange);
  animation: spin 1s ease-in-out infinite;
  margin-right: 8px;
}

@keyframes spin {
  to { transform: rotate(360deg); }
}

.loading-text {
  display: flex;
  align-items: center;
  justify-content: center;
  height: 100px;
  color: #666;
}
```

```
 </style>
</head>
<body>
 <div class="container">
   <div class="header">
     <h1>Counselor 1:1 Meeting Notes</h1>
     <p>Document your biweekly one-on-one meetings with counselors</p>
   </div>

   <div id="successMessage" class="success-message">
     Meeting notes saved successfully!
   </div>

   <div id="errorMessage" class="error-message">
     An error occurred while saving meeting notes. Please try again.
   </div>

   <div class="counselor-selector">
     <div class="form-group">
       <label for="counselorSelect">Select Counselor</label>
       <select id="counselorSelect" onchange="counselorSelected()">
         <option value="">-- Select Counselor --</option>
         <!-- Counselors will be populated here -->
       </select>
     </div>
   </div>

   <!-- Counselor Details Card -->
   <div id="counselorDetailsCard" class="counselor-details-card"
style="display: none;">
     <div class="counselor-details-header">
       <div id="counselorAvatar" class="counselor-avatar">
         <!-- First letter of name will be here -->
```

```
      </div>
      <div class="counselor-info">
        <div id="counselorNameDisplay" class="counselor-name"></div>
        <span id="counselorStatusBadge" class="counselor-status"></span>
      </div>
    </div>

    <div class="counselor-metadata">
      <div class="metadata-item">
        <div class="metadata-label">Email</div>
        <div id="counselorEmailDisplay" class="metadata-value"></div>
      </div>
      <div class="metadata-item">
        <div class="metadata-label">Role</div>
        <div id="counselorRoleDisplay" class="metadata-value"></div>
      </div>
      <div class="metadata-item">
        <div class="metadata-label">Start Date</div>
        <div id="counselorStartDateDisplay" class="metadata-value"></div>
      </div>
    </div>
  </div>

  <div id="meetingContent" style="display: none;">
    <div class="tab-container">
      <div class="tabs">
        <div class="tab active" onclick="showTab('preTab')">Pre-Meeting</div>
        <div class="tab" onclick="showTab('duringTab')">During Meeting</div>
        <div class="tab" onclick="showTab('postTab')">Post-Meeting</div>
      </div>

      <!-- Pre-Meeting Tab -->
      <div id="preTab" class="tab-content active">
```

```
        <div class="card">
          <h3 class="section-header">Pre-Meeting Preparation</h3>

          <div class="form-group">
            <label>Preparation Checklist</label>
            <div>
              <input type="checkbox" id="prepReviewPrevious">
              <label for="prepReviewPrevious">Review previous meeting
notes</label>
            </div>
            <div>
              <input type="checkbox" id="prepCheckMetrics">
              <label for="prepCheckMetrics">Check performance metrics</label>
            </div>
            <div>
              <input type="checkbox" id="prepGatherUpdates">
              <label for="prepGatherUpdates">Gather relevant team/department
updates</label>
            </div>
            <div>
              <input type="checkbox" id="prepDiscussionPoints">
              <label for="prepDiscussionPoints">Prepare preliminary
discussion points</label>
            </div>
          </div>

          <div class="form-group">
            <label>Data Collection</label>
            <div>
              <input type="checkbox" id="dataReviewCalls">
              <label for="dataReviewCalls">Review recent call quality
reports</label>
            </div>
```

```
            <div>
              <input type="checkbox" id="dataCheckIndicators">
              <label for="dataCheckIndicators">Check individual performance
indicators</label>
            </div>
            <div>
              <input type="checkbox" id="dataNoteIssues">
              <label for="dataNoteIssues">Note any team-wide or individual
challenges</label>
            </div>
            <div>
              <input type="checkbox" id="dataIdentifyResources">
              <label for="dataIdentifyResources">Identify potential support
resources</label>
            </div>
          </div>

          <div class="form-group">
            <label for="preparationNotes">Preparation Notes</label>
            <textarea id="preparationNotes" placeholder="Document any key
points to address during the meeting..."></textarea>
          </div>

          <div class="section-actions">
            <button type="button" class="btn btn-secondary"
onclick="google.script.host.close()">Cancel</button>
            <button type="button" class="btn btn-primary"
onclick="showTab('duringTab')">Next: During Meeting</button>
          </div>
        </div>
      </div>

      <!-- During Meeting Tab -->
```

```
      <div id="duringTab" class="tab-content">
        <div class="card">
          <h3 class="section-header">During Meeting (30-Minute Precision
Communication)</h3>

          <!-- Quick Status Sync (5 Minutes) -->
          <h4>1. Quick Status Sync (5 Minutes)</h4>

          <div class="form-group">
            <label for="statusPriority">What's critical to discuss
today?</label>
            <textarea id="statusPriority" placeholder="Document the
counselor's top priorities for this meeting..."></textarea>
          </div>

          <div class="form-group">
            <label>Current Workload Status</label>
            <div class="rating-group">
              <div class="rating-label">Workload Level:</div>
              <div class="rating-options">
                <label><input type="radio" name="workloadStatus"
value="Underutilized"> Underutilized</label>
                <label><input type="radio" name="workloadStatus"
value="Balanced"> Balanced</label>
                <label><input type="radio" name="workloadStatus"
value="Heavy"> Heavy</label>
                <label><input type="radio" name="workloadStatus"
value="Overwhelming"> Overwhelming</label>
              </div>
            </div>
          </div>

          <div class="form-group">
```

```
            <label for="immediateBlockers">Immediate Blockers</label>
            <textarea id="immediateBlockers" placeholder="Document any
immediate obstacles preventing effective work..."></textarea>
          </div>

          <div class="form-group">
            <label>Energy/Motivation Level</label>
            <div class="rating-group">
              <div class="rating-label">Energy Level:</div>
              <div class="rating-options">
                <label><input type="radio" name="energyLevel" value="Low">
Low</label>
                <label><input type="radio" name="energyLevel"
value="Moderate"> Moderate</label>
                <label><input type="radio" name="energyLevel" value="High">
High</label>
                <label><input type="radio" name="energyLevel" value="Very
High"> Very High</label>
              </div>
            </div>
          </div>

          <!-- Performance Deep Dive (10 Minutes) -->
          <h4>2. Performance Deep Dive (10 Minutes)</h4>

          <div class="form-group">
            <label for="quantitativeMetrics">Quantitative Metrics
Review</label>
            <textarea id="quantitativeMetrics" placeholder="Document
observations about call handling, complex case management, quality indicators,
and comparative benchmarks..."></textarea>
          </div>
```

```
            <div class="form-group">
              <label for="qualitativeImpact">Qualitative Impact
Assessment</label>
              <textarea id="qualitativeImpact" placeholder="Document most
meaningful interventions, skill demonstration highlights, areas of exceptional
contribution, and growth opportunities..."></textarea>
            </div>

            <div class="form-group">
              <label for="strategicInsights">Strategic Insight Capture</label>
              <textarea id="strategicInsights" placeholder="Document strengths
to leverage, skill development pathways, and performance trend
analysis..."></textarea>
            </div>

            <!-- Strategic Development Planning (7 Minutes) -->
            <h4>3. Strategic Development Planning (7 Minutes)</h4>

            <div class="form-group">
              <label for="skillMastery">Skill Mastery Mapping</label>
              <textarea id="skillMastery" placeholder="Document current
competency levels, target skill acquisitions, and learning resource
identification..."></textarea>
            </div>

            <div class="form-group">
              <label for="careerTrajectory">Career Trajectory Alignment</label>
              <textarea id="careerTrajectory" placeholder="Document individual
aspirations, organizational needs, and bridging development
gaps..."></textarea>
            </div>

            <div class="form-group">
```

```
            <label for="developmentGoals">Action Development Tracker</label>
            <textarea id="developmentGoals" placeholder="Identify 2-3 specific skill development goals matched with organizational requirements and preliminary learning pathway..."></textarea>
          </div>

          <!-- Support and Resource Optimization (5 Minutes) -->
          <h4>4. Support and Resource Optimization (5 Minutes)</h4>

          <div class="form-group">
            <label for="obstacleIdentification">Obstacle Identification</label>
            <textarea id="obstacleIdentification" placeholder="Document systemic challenges, resource constraints, and potential intervention points..."></textarea>
          </div>

          <div class="form-group">
            <label for="leadershipSupport">Leadership Support Calibration</label>
            <textarea id="leadershipSupport" placeholder="Document immediate support needs, potential mentorship opportunities, and training/resource recommendations..."></textarea>
          </div>

          <!-- Commitment and Momentum Generation (3 Minutes) -->
          <h4>5. Commitment and Momentum Generation (3 Minutes)</h4>

          <div class="form-group">
            <label for="commitmentSynthesis">Commitments Synthesis</label>
            <textarea id="commitmentSynthesis" placeholder="Document clearly articulated action items, mutual accountability establishment, and next check-in preparation..."></textarea>
```

```
            </div>

            <div class="form-group">
              <label>Momentum Acceleration Checklist</label>
              <div>
                <input type="checkbox" id="momentumActionItems">
                <label for="momentumActionItems">Confirmed 3 specific action
items</label>
              </div>
              <div>
                <input type="checkbox" id="momentumOwnership">
                <label for="momentumOwnership">Established clear
ownership</label>
              </div>
              <div>
                <input type="checkbox" id="momentumFollowUp">
                <label for="momentumFollowUp">Set precise follow-up
expectations</label>
              </div>
            </div>

            <div class="section-actions">
              <button type="button" class="btn btn-secondary"
onclick="showTab('preTab')">Back: Pre-Meeting</button>
              <button type="button" class="btn btn-primary"
onclick="showTab('postTab')">Next: Post-Meeting</button>
            </div>
          </div>
        </div>

        <!-- Post-Meeting Tab -->
        <div id="postTab" class="tab-content">
          <div class="card">
```

```
            <h3 class="section-header">Post-Meeting Follow-Up Protocol</h3>

            <div class="form-group">
              <label>Immediate Documentation</label>
              <div>
                <input type="checkbox" id="docFinalizeNotes">
                <label for="docFinalizeNotes">Finalize meeting notes</label>
              </div>
              <div>
                <input type="checkbox" id="docClarifyPoints">
                <label for="docClarifyPoints">Clarify and expand on key
discussion points</label>
              </div>
              <div>
                <input type="checkbox" id="docActionItems">
                <label for="docActionItems">Document action items with clear
ownership</label>
              </div>
              <div>
                <input type="checkbox" id="docFollowUpResources">
                <label for="docFollowUpResources">Note any required follow-up
resources</label>
              </div>
            </div>

            <div class="form-group">
              <label for="actionItems">Action Item Tracking</label>
              <div id="actionItemsContainer">
                <div class="action-item">
                  <div class="form-group">
                    <label>Action Item 1</label>
                    <input type="text" id="actionItem1"
placeholder="Description of action">
```

```
                </div>
                <div class="form-group">
                  <label>Owner</label>
                  <select id="actionOwner1">
                    <option value="Counselor">Counselor</option>
                    <option value="Team Lead">Team Lead</option>
                    <option value="Both">Both</option>
                  </select>
                </div>
                <div class="form-group">
                  <label>Target Completion</label>
                  <input type="date" id="actionDate1">
                </div>
              </div>
              <!-- Additional action items will be added here -->
            </div>
            <button type="button" class="btn btn-secondary btn-sm"
onclick="addActionItem()">+ Add Action Item</button>
          </div>

          <div class="form-group">
            <label>Continued Support Mechanisms</label>
            <div>
              <input type="checkbox" id="supportFollowUp">
              <label for="supportFollowUp">Schedule follow-up check-in
points</label>
            </div>
            <div>
              <input type="checkbox" id="supportMentorship">
              <label for="supportMentorship">Identify potential mentorship or
training opportunities</label>
            </div>
            <div>
```

```
                <input type="checkbox" id="supportResources">
                <label for="supportResources">Connect to additional resources
if needed</label>
              </div>
            </div>

            <div class="form-group">
              <label>Reflection and Improvement</label>
              <div class="form-group">
                <label for="reflectionSupport">What support did the counselor
most need?</label>
                <textarea id="reflectionSupport" placeholder="Document insights
about counselor support needs..."></textarea>
              </div>
              <div class="form-group">
                <label for="reflectionFacilitation">How can I better facilitate
their growth?</label>
                <textarea id="reflectionFacilitation" placeholder="Document
ways to improve your coaching approach..."></textarea>
              </div>
              <div class="form-group">
                <label for="reflectionMechanisms">Are there systemic support
mechanisms we can improve?</label>
                <textarea id="reflectionMechanisms" placeholder="Document
potential improvements to systems and processes..."></textarea>
              </div>
            </div>

            <div class="form-group">
              <label>Confidential Development Notes</label>
              <div class="form-group">
                <label for="devGrowthAreas">Discrete observations about
potential growth areas</label>
```

```
              <textarea id="devGrowthAreas" placeholder="Document private
observations about growth areas..."></textarea>
            </div>
            <div class="form-group">
              <label for="devStrengths">Strengths to further develop</label>
              <textarea id="devStrengths" placeholder="Document strengths
that can be further developed..."></textarea>
            </div>
            <div class="form-group">
              <label for="devCareer">Long-term career development
insights</label>
              <textarea id="devCareer" placeholder="Document insights about
long-term career development..."></textarea>
            </div>
          </div>

          <div class="form-group">
            <label for="meetingDate">Meeting Date</label>
            <input type="date" id="meetingDate">
          </div>

          <div class="form-group">
            <label for="nextCheckInDate">Next Check-In Date</label>
            <input type="date" id="nextCheckInDate">
          </div>

          <div class="section-actions">
            <button type="button" class="btn btn-secondary"
onclick="showTab('duringTab')">Back: During Meeting</button>
            <button type="button" class="btn btn-primary"
onclick="saveOneOnOneNotes()">Save Meeting Notes</button>
          </div>
        </div>
```

```
      </div>
    </div>
  </div>

  <div id="previousMeetingsSection" style="display: none;" class="card
previous-meetings">
    <h3>Previous 1:1 Meetings</h3>
    <div id="previousMeetingsList">
      <div class="loading-text">
        <div class="loading-spinner"></div>
        Loading previous meetings...
      </div>
    </div>
  </div>
 </div>
  <script>
  // Initialize date fields
  document.addEventListener('DOMContentLoaded', function() {
    const today = new Date().toISOString().split('T')[0];
    document.getElementById('meetingDate').value = today;

    // Calculate default next check-in (two weeks from today)
    const nextCheckIn = new Date();
    nextCheckIn.setDate(nextCheckIn.getDate() + 14);
    document.getElementById('nextCheckInDate').value =
nextCheckIn.toISOString().split('T')[0];

    // Load team members from Counselor Tracking sheet
    loadCounselors();
  });

  // Load counselors from the server
  function loadCounselors() {
```

```
    google.script.run
      .withSuccessHandler(populateCounselorDropdown)
      .withFailureHandler(function(error) {
        console.error('Error loading team members:', error);
        document.getElementById('errorMessage').style.display = 'block';
        document.getElementById('errorMessage').textContent = 'Could not load
team members. Please refresh the page.';
      })
      .getTeamMembers();
  }

  // Populate counselor dropdown
  function populateCounselorDropdown(teamMembers) {
    const select = document.getElementById('counselorSelect');

    // Clear existing options (except the first one)
    while (select.options.length > 1) {
      select.remove(1);
    }

    // Filter for valid counselors and sort alphabetically by name
    const validMembers = teamMembers.filter(member =>
      member.name && member.email && (
        member.role.toLowerCase().includes('counselor') ||
        member.status.toLowerCase() === 'active'
      )
    );

    validMembers.sort((a, b) => a.name.localeCompare(b.name));

    // Add team member options
    if (validMembers.length > 0) {
      validMembers.forEach(member => {
```

```
        const option = document.createElement('option');
        option.value = member.email; // Use email as unique identifier
        option.textContent = member.name;

        // Add status indicator if not active
        if (member.status && member.status.toLowerCase() !== 'active') {
          option.textContent += ` (${member.status})`;
        }

        // Store additional data for later use
        option.dataset.status = member.status || 'Active';
        option.dataset.role = member.role || 'Counselor';
        option.dataset.startDate = member.startDate || '';

        select.appendChild(option);
      });
    } else {
      // Add placeholder option if no valid members
      const option = document.createElement('option');
      option.value = '';
      option.textContent = 'No counselors found';
      option.disabled = true;
      select.appendChild(option);
    }
}

// Counselor selection handler
function counselorSelected() {
  const select = document.getElementById('counselorSelect');
  const counselorEmail = select.value;

  // Reset and hide sections
  document.getElementById('counselorDetailsCard').style.display = 'none';
```

```
    document.getElementById('meetingContent').style.display = 'none';
    document.getElementById('previousMeetingsSection').style.display =
'none';

    // Clear previous error/success messages
    document.getElementById('successMessage').style.display = 'none';
    document.getElementById('errorMessage').style.display = 'none';

    if (!counselorEmail) {
      return; // Nothing selected
    }

    // Get selected option data
    const selectedOption = select.options[select.selectedIndex];
    const counselorName = selectedOption.textContent.split(' (')[0]; //
Remove status if present
    const counselorStatus = selectedOption.dataset.status || 'Active';
    const counselorRole = selectedOption.dataset.role || 'Counselor';
    const counselorStartDate = selectedOption.dataset.startDate || '';

    // Display counselor details
    displayCounselorDetails({
      name: counselorName,
      email: counselorEmail,
      status: counselorStatus,
      role: counselorRole,
      startDate: counselorStartDate
    });

    // Show meeting content
    document.getElementById('meetingContent').style.display = 'block';

    // Load previous meetings
```

```
    loadPreviousMeetings(counselorEmail);
  }

  // Display counselor details
  function displayCounselorDetails(member) {
    // Show counselor details card
    const detailsCard = document.getElementById('counselorDetailsCard');
    detailsCard.style.display = 'block';

    // Avatar (first letter of name)
    const avatar = document.getElementById('counselorAvatar');
    avatar.textContent = member.name ? member.name.charAt(0).toUpperCase() :
'?';

    // Name
    document.getElementById('counselorNameDisplay').textContent =
member.name;

    // Status Badge
    const statusBadge = document.getElementById('counselorStatusBadge');
    statusBadge.textContent = member.status;
    statusBadge.className = `counselor-status
status-${member.status.toLowerCase()}`;

    // Email
    document.getElementById('counselorEmailDisplay').textContent =
member.email;

    // Role
    document.getElementById('counselorRoleDisplay').textContent = member.role
|| 'Counselor';

    // Start Date
```

```
    document.getElementById('counselorStartDateDisplay').textContent =
      member.startDate ? new Date(member.startDate).toLocaleDateString() :
'Not available';
  }

  // Load previous meetings
  function loadPreviousMeetings(counselorEmail) {
    const section = document.getElementById('previousMeetingsSection');
    section.style.display = 'block';

    const container = document.getElementById('previousMeetingsList');
    container.innerHTML = '<div class="loading-text"><div
class="loading-spinner"></div>Loading previous meetings...</div>';

    google.script.run
      .withSuccessHandler(displayPreviousMeetings)
      .withFailureHandler(function(error) {
        console.error('Error loading previous meetings:', error);
        container.innerHTML = '<p class="center">Error loading previous
meetings. Please try again.</p>';
      })
      .getOneOnOneMeetings(counselorEmail);
  }

  // Display previous meetings
  function displayPreviousMeetings(meetings) {
    const container = document.getElementById('previousMeetingsList');

    // Clear loading indicator
    container.innerHTML = '';

    // Show or hide section based on meetings
    if (meetings && meetings.length > 0) {
```

```
        // Sort meetings by date (newest first)
        meetings.sort((a, b) => new Date(b.meetingDate) - new
Date(a.meetingDate));

        // Add meetings to the list
        meetings.forEach(meeting => {
          const meetingEl = document.createElement('div');
          meetingEl.className = 'meeting-item';
          meetingEl.onclick = function() { loadMeetingDetails(meeting.id); };

          const dateEl = document.createElement('div');
          dateEl.className = 'meeting-date';
          dateEl.textContent = new
Date(meeting.meetingDate).toLocaleDateString();

          const summaryEl = document.createElement('div');
          summaryEl.className = 'meeting-summary';
          summaryEl.textContent = meeting.summary || 'No summary available';

          const nextCheckInEl = document.createElement('div');
          nextCheckInEl.className = 'meeting-next-checkin';
          nextCheckInEl.textContent = meeting.nextCheckInDate
            ? `Next Check-In: ${new
Date(meeting.nextCheckInDate).toLocaleDateString()}`
            : '';

          meetingEl.appendChild(dateEl);
          meetingEl.appendChild(summaryEl);
          if (meeting.nextCheckInDate) {
            meetingEl.appendChild(nextCheckInEl);
          }
          container.appendChild(meetingEl);
        });
```

```
    } else {
      // Show message if no meetings
      container.innerHTML = '<p class="center">No previous meetings found.</p>';
    }
  }

  // Load meeting details
  function loadMeetingDetails(meetingId) {
    // Show loading state in previous meetings list
    document.getElementById('previousMeetingsList').innerHTML =
      '<div class="loading-text"><div class="loading-spinner"></div>Loading meeting details...</div>';

    google.script.run
      .withSuccessHandler(populateMeetingDetails)
      .withFailureHandler(function(error) {
        console.error('Error loading meeting details:', error);
        document.getElementById('errorMessage').style.display = 'block';
        document.getElementById('errorMessage').textContent = 'Could not load meeting details.';
        // Reload the meetings list
        const counselorEmail = document.getElementById('counselorSelect').value;
        loadPreviousMeetings(counselorEmail);
      })
      .getOneOnOneMeetingDetails(meetingId);
  }

  // Populate meeting details into the form
  function populateMeetingDetails(meeting) {
    if (!meeting) {
      document.getElementById('errorMessage').style.display = 'block';
```

```
        document.getElementById('errorMessage').textContent = 'Meeting details not found.';
        const counselorEmail = document.getElementById('counselorSelect').value;
        loadPreviousMeetings(counselorEmail);
        return;
      }

      // Set meeting date
      document.getElementById('meetingDate').value = formatDate(meeting.meetingDate);
      document.getElementById('nextCheckInDate').value = formatDate(meeting.nextCheckInDate);

      // Populate pre-meeting tab
      document.getElementById('prepReviewPrevious').checked = meeting.prepReviewPrevious || false;
      document.getElementById('prepCheckMetrics').checked = meeting.prepCheckMetrics || false;
      document.getElementById('prepGatherUpdates').checked = meeting.prepGatherUpdates || false;
      document.getElementById('prepDiscussionPoints').checked = meeting.prepDiscussionPoints || false;
      document.getElementById('dataReviewCalls').checked = meeting.dataReviewCalls || false;
      document.getElementById('dataCheckIndicators').checked = meeting.dataCheckIndicators || false;
      document.getElementById('dataNoteIssues').checked = meeting.dataNoteIssues || false;
      document.getElementById('dataIdentifyResources').checked = meeting.dataIdentifyResources || false;
      document.getElementById('preparationNotes').value = meeting.preparationNotes || '';
```

```
    // Populate during-meeting tab
    document.getElementById('statusPriority').value = meeting.statusPriority
|| '';
    setRadioValue('workloadStatus', meeting.workloadStatus);
    document.getElementById('immediateBlockers').value =
meeting.immediateBlockers || '';
    setRadioValue('energyLevel', meeting.energyLevel);
    document.getElementById('quantitativeMetrics').value =
meeting.quantitativeMetrics || '';
    document.getElementById('qualitativeImpact').value =
meeting.qualitativeImpact || '';
    document.getElementById('strategicInsights').value =
meeting.strategicInsights || '';
    document.getElementById('skillMastery').value = meeting.skillMastery ||
'';
    document.getElementById('careerTrajectory').value =
meeting.careerTrajectory || '';
    document.getElementById('developmentGoals').value =
meeting.developmentGoals || '';
    document.getElementById('obstacleIdentification').value =
meeting.obstacleIdentification || '';
    document.getElementById('leadershipSupport').value =
meeting.leadershipSupport || '';
    document.getElementById('commitmentSynthesis').value =
meeting.commitmentSynthesis || '';
    document.getElementById('momentumActionItems').checked =
meeting.momentumActionItems || false;
    document.getElementById('momentumOwnership').checked =
meeting.momentumOwnership || false;
    document.getElementById('momentumFollowUp').checked =
meeting.momentumFollowUp || false;
```

```
    // Populate post-meeting tab
    document.getElementById('docFinalizeNotes').checked =
meeting.docFinalizeNotes || false;
    document.getElementById('docClarifyPoints').checked =
meeting.docClarifyPoints || false;
    document.getElementById('docActionItems').checked =
meeting.docActionItems || false;
    document.getElementById('docFollowUpResources').checked =
meeting.docFollowUpResources || false;

    // Set action items
    const actionItemsContainer =
document.getElementById('actionItemsContainer');
    actionItemsContainer.innerHTML = ''; // Clear existing items

    if (meeting.actionItems && meeting.actionItems.length > 0) {
      meeting.actionItems.forEach((item, index) => {
        addActionItem(item);
      });
    } else {
      // Add a default empty action item
      addActionItem();
    }

    // Set additional fields
    document.getElementById('supportFollowUp').checked =
meeting.supportFollowUp || false;
    document.getElementById('supportMentorship').checked =
meeting.supportMentorship || false;
    document.getElementById('supportResources').checked =
meeting.supportResources || false;
    document.getElementById('reflectionSupport').value =
meeting.reflectionSupport || '';
```

```
    document.getElementById('reflectionFacilitation').value =
meeting.reflectionFacilitation || '';
    document.getElementById('reflectionMechanisms').value =
meeting.reflectionMechanisms || '';
    document.getElementById('devGrowthAreas').value = meeting.devGrowthAreas
|| '';
    document.getElementById('devStrengths').value = meeting.devStrengths ||
'';
    document.getElementById('devCareer').value = meeting.devCareer || '';

    // Show the first tab
    showTab('preTab');

    // Reload the meetings list to remove loading indicator
    const counselorEmail = document.getElementById('counselorSelect').value;
    loadPreviousMeetings(counselorEmail);
  }

  // Format date for input field (YYYY-MM-DD)
  function formatDate(dateString) {
    if (!dateString) return '';
    const date = new Date(dateString);
    if (isNaN(date.getTime())) return '';
    return date.toISOString().split('T')[0];
  }

  // Show the selected tab
  function showTab(tabId) {
    // Hide all tabs
    document.querySelectorAll('.tab-content').forEach(tab => {
      tab.classList.remove('active');
    });
```

```
  // Show the selected tab
  document.getElementById(tabId).classList.add('active');

  // Update tab buttons
  document.querySelectorAll('.tab').forEach(tab => {
    tab.classList.remove('active');
  });

  // Highlight the active tab button
  const tabIndex = ['preTab', 'duringTab', 'postTab'].indexOf(tabId);
  document.querySelectorAll('.tab')[tabIndex].classList.add('active');
}

// Helper function to set radio button value
function setRadioValue(name, value) {
  if (!value) return;

  const radios = document.getElementsByName(name);
  for (let i = 0; i < radios.length; i++) {
    if (radios[i].value === value) {
      radios[i].checked = true;
      break;
    }
  }
}

// Helper function to get radio button value
function getRadioValue(name) {
  const radios = document.getElementsByName(name);
  for (let i = 0; i < radios.length; i++) {
    if (radios[i].checked) {
      return radios[i].value;
    }
```

```
    }
    return null;
  }

  // Add action item to the form
  function addActionItem(item = null) {
    const container = document.getElementById('actionItemsContainer');
    const index = container.children.length + 1;

    const actionItem = document.createElement('div');
    actionItem.className = 'action-item';
    actionItem.innerHTML = `
      <div class="form-group">
        <label>Action Item ${index}</label>
        <input type="text" id="actionItem${index}" placeholder="Description
of action" value="${item ? item.description : ''}">
      </div>
      <div class="form-group">
        <label>Owner</label>
        <select id="actionOwner${index}">
          <option value="Counselor" ${item && item.owner === 'Counselor' ?
'selected' : ''}>Counselor</option>
          <option value="Team Lead" ${item && item.owner === 'Team Lead' ?
'selected' : ''}>Team Lead</option>
          <option value="Both" ${item && item.owner === 'Both' ? 'selected' :
''}>Both</option>
        </select>
      </div>
      <div class="form-group">
        <label>Target Completion</label>
        <input type="date" id="actionDate${index}" value="${item && item.date
? formatDate(item.date) : ''}">
      </div>
```

```
      ${index > 1 ? `<button type="button" class="btn btn-secondary"
style="margin-top: 10px;" onclick="removeActionItem(this)">Remove</button>` :
''}
    `;

    container.appendChild(actionItem);
  }

  // Remove an action item
  function removeActionItem(button) {
    const actionItem = button.parentNode;
    actionItem.parentNode.removeChild(actionItem);

    // Renumber remaining action items
    const container = document.getElementById('actionItemsContainer');
    const items = container.getElementsByClassName('action-item');

    for (let i = 0; i < items.length; i++) {
      const labels = items[i].getElementsByTagName('label');
      labels[0].textContent = `Action Item ${i + 1}`;

      const inputs = items[i].getElementsByTagName('input');
      inputs[0].id = `actionItem${i + 1}`;
      inputs[1].id = `actionDate${i + 1}`;

      const selects = items[i].getElementsByTagName('select');
      selects[0].id = `actionOwner${i + 1}`;
    }
  }

  // Save one-on-one meeting notes
  function saveOneOnOneNotes() {
    // Get counselor data
```

```
    const counselorSelect = document.getElementById('counselorSelect');
    const selectedOption = counselorSelect.options[counselorSelect.selectedIndex];

    if (!counselorSelect.value) {
      alert('Please select a counselor');
      return;
    }

    // Gather action items
    const actionItems = [];
    const actionItemsContainer = document.getElementById('actionItemsContainer');
    const actionElements = actionItemsContainer.getElementsByClassName('action-item');

    for (let i = 0; i < actionElements.length; i++) {
      const index = i + 1;
      const description = document.getElementById(`actionItem${index}`).value;

      if (description) {
        actionItems.push({
          description: description,
          owner: document.getElementById(`actionOwner${index}`).value,
          date: document.getElementById(`actionDate${index}`).value
        });
      }
    }

    // Generate a summary based on the status priority
    let summary = document.getElementById('statusPriority').value.trim();
    if (summary.length > 100) {
```

```
      summary = summary.substring(0, 97) + '...';
    } else if (!summary) {
      summary = 'Regular 1:1 check-in';
    }

    // Prepare meeting data
    const meetingData = {
      counselorName: selectedOption.textContent.split(' (')[0], // Remove
status if present
      counselorEmail: counselorSelect.value,
      meetingDate: document.getElementById('meetingDate').value,
      nextCheckInDate: document.getElementById('nextCheckInDate').value,
      summary: summary,

      // Pre-meeting data
      prepReviewPrevious:
document.getElementById('prepReviewPrevious').checked,
      prepCheckMetrics: document.getElementById('prepCheckMetrics').checked,
      prepGatherUpdates:
document.getElementById('prepGatherUpdates').checked,
      prepDiscussionPoints:
document.getElementById('prepDiscussionPoints').checked,
      dataReviewCalls: document.getElementById('dataReviewCalls').checked,
      dataCheckIndicators:
document.getElementById('dataCheckIndicators').checked,
      dataNoteIssues: document.getElementById('dataNoteIssues').checked,
      dataIdentifyResources:
document.getElementById('dataIdentifyResources').checked,
      preparationNotes: document.getElementById('preparationNotes').value,

      // During-meeting data
      statusPriority: document.getElementById('statusPriority').value,
      workloadStatus: getRadioValue('workloadStatus'),
```

```
        immediateBlockers: document.getElementById('immediateBlockers').value,
        energyLevel: getRadioValue('energyLevel'),
        quantitativeMetrics:
document.getElementById('quantitativeMetrics').value,
        qualitativeImpact: document.getElementById('qualitativeImpact').value,
        strategicInsights: document.getElementById('strategicInsights').value,
        skillMastery: document.getElementById('skillMastery').value,
        careerTrajectory: document.getElementById('careerTrajectory').value,
        developmentGoals: document.getElementById('developmentGoals').value,
        obstacleIdentification:
document.getElementById('obstacleIdentification').value,
        leadershipSupport: document.getElementById('leadershipSupport').value,
        commitmentSynthesis:
document.getElementById('commitmentSynthesis').value,
        momentumActionItems:
document.getElementById('momentumActionItems').checked,
        momentumOwnership:
document.getElementById('momentumOwnership').checked,
        momentumFollowUp: document.getElementById('momentumFollowUp').checked,

        // Post-meeting data
        docFinalizeNotes: document.getElementById('docFinalizeNotes').checked,
        docClarifyPoints: document.getElementById('docClarifyPoints').checked,
        docActionItems: document.getElementById('docActionItems').checked,
        docFollowUpResources:
document.getElementById('docFollowUpResources').checked,
        actionItems: actionItems,
        supportFollowUp: document.getElementById('supportFollowUp').checked,
        supportMentorship:
document.getElementById('supportMentorship').checked,
        supportResources: document.getElementById('supportResources').checked,
        reflectionSupport: document.getElementById('reflectionSupport').value,
```

```
      reflectionFacilitation:
document.getElementById('reflectionFacilitation').value,
      reflectionMechanisms:
document.getElementById('reflectionMechanisms').value,
      devGrowthAreas: document.getElementById('devGrowthAreas').value,
      devStrengths: document.getElementById('devStrengths').value,
      devCareer: document.getElementById('devCareer').value
    };

    // Show loading state
    const submitButton = document.querySelector('#postTab .btn-primary');
    const originalText = submitButton.textContent;
    submitButton.innerHTML = '<div class="loading-spinner"></div> Saving...';
    submitButton.disabled = true;

    // Clear previous messages
    document.getElementById('successMessage').style.display = 'none';
    document.getElementById('errorMessage').style.display = 'none';

    // Submit to the server
    google.script.run
      .withSuccessHandler(function(result) {
        // Reset button state
        submitButton.textContent = originalText;
        submitButton.disabled = false;

        if (result.success) {
          // Show success message
          document.getElementById('successMessage').style.display = 'block';
          document.getElementById('successMessage').textContent =
result.message || 'Meeting notes saved successfully!';

          // Ask if user wants to send follow-up email
```

```
          const sendEmail = confirm('Meeting notes saved successfully. Would you like to send a follow-up email to the counselor?');

          if (sendEmail) {
            // Send follow-up email
            sendFollowUpEmail(meetingData);
          } else {
            // Reload previous meetings
            loadPreviousMeetings(meetingData.counselorEmail);
          }
        } else {
          // Show error message
          document.getElementById('errorMessage').style.display = 'block';
          document.getElementById('errorMessage').textContent = result.message || 'An error occurred while saving meeting notes. Please try again.';
        }
      })
      .withFailureHandler(function(error) {
        // Reset button state
        submitButton.textContent = originalText;
        submitButton.disabled = false;

        // Show error message
        document.getElementById('errorMessage').style.display = 'block';
        document.getElementById('errorMessage').textContent = error.message || 'An error occurred while saving meeting notes. Please try again.';
      })
      .saveOneOnOneNotes(meetingData);
  }

  // Send follow-up email to counselor
  function sendFollowUpEmail(meetingData) {
```

```
    google.script.run
      .withSuccessHandler(function(result) {
        if (result.success) {
          alert('Follow-up email sent successfully.');
        } else {
          alert('Error sending follow-up email: ' + result.message);
        }

        // Reload previous meetings
        loadPreviousMeetings(meetingData.counselorEmail);
      })
      .withFailureHandler(function(error) {
        alert('Error sending follow-up email: ' + error.message);

        // Reload previous meetings
        loadPreviousMeetings(meetingData.counselorEmail);
      })
      .sendOneOnOneFollowUpEmail(meetingData);
  }
 </script>
</body>
</html>
<!DOCTYPE html>
<html>
<head>
 <base target="_top">
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <link
href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500;700&fam
ily=Poppins:wght@500;600;700&display=swap" rel="stylesheet">
 <style>
   /* Variables for Trevor Project color scheme */
```

```css
:root {
  /* Primary Colors */
  --trevor-orange: #FF786E;
  --deep-blue: #001A4E;
  --purple: #9A3499;
  --teal: #137F6A;

  /* Secondary Colors */
  --light-blue: #4F52DE;
  --soft-yellow: #FFAD8D;
  --lavender: #B3AE4A;
  --light-purple: #F54AC;

  /* Tertiary Colors */
  --soft-green: #BAE2CE;
  --pale-yellow: #FFF2DF;
  --light-pink: #FBCBBE;
  --soft-lavender: #D1CFCC;

  /* Gradients */
  --primary-gradient: linear-gradient(135deg, var(--trevor-orange), var(--light-pink));
  --calm-gradient: linear-gradient(135deg, var(--light-blue), var(--soft-green));
  --support-gradient: linear-gradient(45deg, var(--purple), var(--light-purple));
  --background-gradient: linear-gradient(180deg, var(--pale-yellow), #FFFFFF);

  /* Spacing */
  --spacing-xs: 4px;
  --spacing-sm: 8px;
  --spacing-md: 16px;
```

```
  --spacing-lg: 24px;
  --spacing-xl: 32px;

  /* Borders */
  --border-radius-sm: 8px;
  --border-radius-md: 12px;
  --border-radius-lg: 16px;
}

body {
  font-family: 'Roboto', sans-serif;
  margin: 0;
  padding: 0;
  background-color: var(--pale-yellow);
  color: var(--deep-blue);
}

.container {
  max-width: 900px;
  margin: 0 auto;
  padding: var(--spacing-md);
}

h1, h2, h3, h4 {
  font-family: 'Poppins', sans-serif;
  color: var(--deep-blue);
}

.header {
  margin-bottom: var(--spacing-lg);
  text-align: center;
}
```

```css
.card {
  background: white;
  border-radius: var(--border-radius-md);
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.05);
  padding: var(--spacing-lg);
  margin-bottom: var(--spacing-lg);
  position: relative;
}

.card::before {
  content: '';
  position: absolute;
  left: 0;
  top: 0;
  height: 100%;
  width: 3px;
  background: var(--light-blue);
}

.team-summary {
  display: flex;
  justify-content: space-around;
  align-items: center;
  margin-bottom: var(--spacing-lg);
}

.summary-metric {
  text-align: center;
}

.metric-value {
  font-size: 36px;
  font-weight: 700;
```

```css
  margin-bottom: var(--spacing-xs);
}

.metric-label {
  font-size: 14px;
  color: #666;
}

.counselor-card {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: var(--spacing-md);
  border-radius: var(--border-radius-sm);
  background: white;
  margin-bottom: var(--spacing-sm);
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.05);
  transition: all 0.2s ease;
}

.counselor-card:hover {
  transform: translateY(-2px);
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
}

.counselor-info {
  flex: 1;
}

.counselor-name {
  font-weight: 500;
  margin-bottom: var(--spacing-xs);
}
```

```css
.counselor-meta {
  display: flex;
  color: #666;
  font-size: 12px;
}

.meta-item {
  margin-right: var(--spacing-md);
}

.counselor-score {
  font-size: 24px;
  font-weight: 700;
  color: var(--deep-blue);
  padding: 0 var(--spacing-md);
}

.score-badge {
  display: inline-block;
  width: 12px;
  height: 12px;
  border-radius: 50%;
  margin-right: var(--spacing-xs);
}

.badge-excellent {
  background-color: var(--soft-green);
}

.badge-good {
  background-color: var(--soft-yellow);
}
```

```css
.badge-needs-improvement {
  background-color: var(--light-pink);
}

.trend-icon {
  margin-left: var(--spacing-xs);
  font-size: 14px;
}

.trend-up {
  color: var(--soft-green);
}

.trend-down {
  color: var(--light-pink);
}

.trend-neutral {
  color: #999;
}

.filters {
  display: flex;
  justify-content: space-between;
  margin-bottom: var(--spacing-md);
}

.filter-group {
  flex: 1;
  margin-right: var(--spacing-md);
}
```

```css
.filter-group:last-child {
  margin-right: 0;
}

.filter-label {
  font-size: 12px;
  margin-bottom: var(--spacing-xs);
  display: block;
}

.filter-select {
  width: 100%;
  padding: var(--spacing-xs) var(--spacing-sm);
  border: 1px solid #ddd;
  border-radius: var(--border-radius-sm);
}

.btn {
  border: none;
  padding: var(--spacing-sm) var(--spacing-md);
  border-radius: var(--border-radius-sm);
  font-weight: 500;
  cursor: pointer;
  transition: all 0.2s ease;
}

.btn-primary {
  background: var(--primary-gradient);
  color: white;
}

.btn-primary:hover {
  box-shadow: 0 4px 8px rgba(255, 120, 110, 0.3);
```

```
  transform: translateY(-2px);
}

.btn-secondary {
  background: white;
  color: var(--deep-blue);
  border: 1px solid #ddd;
}

.btn-secondary:hover {
  background: #f5f5f5;
}

.review-details {
  display: none;
  padding: var(--spacing-md);
  background-color: #f9f9f9;
  border-radius: var(--border-radius-sm);
  margin-top: var(--spacing-sm);
}

.detail-section {
  margin-bottom: var(--spacing-md);
}

.detail-section h4 {
  margin-bottom: var(--spacing-sm);
  font-size: 16px;
}

.detail-item {
  display: flex;
  justify-content: space-between;
```

```
  padding: var(--spacing-xs) 0;
  border-bottom: 1px solid #eee;
}

.detail-label {
  font-weight: 500;
}

.detail-value {
  color: #666;
}

.tabs {
  display: flex;
  margin-bottom: var(--spacing-md);
  border-bottom: 1px solid #ddd;
}

.tab-item {
  padding: var(--spacing-sm) var(--spacing-lg);
  cursor: pointer;
  border-bottom: 2px solid transparent;
  transition: all 0.2s ease;
}

.tab-item.active {
  border-bottom-color: var(--trevor-orange);
  font-weight: 500;
}

.tab-content {
  display: none;
}
```

```css
.tab-content.active {
  display: block;
}

.error-message {
  background-color: var(--light-pink);
  border-radius: var(--border-radius-sm);
  padding: var(--spacing-md);
  margin-bottom: var(--spacing-md);
  display: none;
}

.loading {
  text-align: center;
  padding: var(--spacing-lg);
  color: #666;
}

/* Responsive adjustments */
@media (max-width: 768px) {
  .team-summary {
    flex-direction: column;
  }

  .summary-metric {
    margin-bottom: var(--spacing-md);
  }

  .filters {
    flex-direction: column;
  }
```

```
    .filter-group {
      margin-right: 0;
      margin-bottom: var(--spacing-sm);
    }
  }
 </style>
</head>
<body>
 <div class="container">
  <div class="header">
    <h1>Quality Reports</h1>
    <p>Monitor quality metrics and counselor performance</p>
  </div>

  <!-- Reviews List Tab -->
  <div id="reviews-list" class="tab-content">
    <div class="filters">
      <div class="filter-group">
        <label class="filter-label">Sort By</label>
        <select id="sortSelect" class="filter-select">
          <option value="date-desc">Date (Newest First)</option>
          <option value="date-asc">Date (Oldest First)</option>
          <option value="score-desc">Score (Highest First)</option>
          <option value="score-asc">Score (Lowest First)</option>
        </select>
      </div>

      <div class="filter-group">
        <label class="filter-label">Filter By Counselor</label>
        <select id="filterCounselorSelect" class="filter-select">
          <option value="">All Counselors</option>
          <!-- Counselors will be populated here -->
        </select>
```

```
      </div>

      <div class="filter-group">
        <label class="filter-label">Score Range</label>
        <select id="scoreRangeSelect" class="filter-select">
          <option value="">All Scores</option>
          <option value="90-100">90-100% (Excellent)</option>
          <option value="75-89">75-89% (Good)</option>
          <option value="0-74">Below 75% (Needs Improvement)</option>
        </select>
      </div>
    </div>

    <div id="reviewsListLoading" class="loading">
      Loading reviews...
    </div>

    <div id="reviewsListContent" style="display: none;">
      <div class="card">
        <h3>Quality Reviews</h3>

        <div id="reviewsList">
          <!-- Review items will be added here -->
        </div>
      </div>
    </div>
  </div>
</div>
 <script>
  // Global variables
  let allReviews = [];
  let allCounselors = [];
  let teamStats = null;
```

```
// Initialize the reports
function initialize() {
  setupTabs();
  loadTeamOverview();
  loadCounselors();
  loadAllReviews();
}

// Set up tab switching
function setupTabs() {
  const tabs = document.querySelectorAll('.tab-item');

  tabs.forEach(function(tab) {
    tab.addEventListener('click', function() {
      // Remove active class from all tabs
      tabs.forEach(function(t) {
        t.classList.remove('active');
      });

      // Add active class to clicked tab
      tab.classList.add('active');

      // Hide all tab content
      const tabContents = document.querySelectorAll('.tab-content');
      tabContents.forEach(function(content) {
        content.classList.remove('active');
      });

      // Show selected tab content
      const tabId = tab.getAttribute('data-tab');
      document.getElementById(tabId).classList.add('active');
    });
```

```
    });
  }

  // Load team overview data
  function loadTeamOverview() {
    document.getElementById('teamOverviewLoading').style.display = 'block';
    document.getElementById('teamOverviewContent').style.display = 'none';

    google.script.run
      .withSuccessHandler(function(result) {
        teamStats = result;
        renderTeamOverview(result);

        document.getElementById('teamOverviewLoading').style.display =
'none';
        document.getElementById('teamOverviewContent').style.display =
'block';
      })
      .withFailureHandler(function(error) {
        showError("Failed to load team overview: " + error);
        document.getElementById('teamOverviewLoading').style.display =
'none';
      })
      .getQualityStatsByTeam();
  }

  // Render team overview
  function renderTeamOverview(data) {
    // Update summary metrics
    document.getElementById('teamAverageScore').textContent =
data.teamAverage + '%';
    document.getElementById('totalReviews').textContent =
data.counselors.reduce((sum, c) => sum + c.count, 0);
```

```
    document.getElementById('counselorsReviewed').textContent =
data.counselors.length;

    // Sort counselors by average score (highest first)
    const sortedCounselors = [...data.counselors].sort((a, b) =>
b.averageScore - a.averageScore);

    // Render counselor list
    const counselorListElement = document.getElementById('counselorList');
    counselorListElement.innerHTML = '';

    sortedCounselors.forEach(function(counselor) {
      const counselorCard = document.createElement('div');
      counselorCard.className = 'counselor-card';

      // Determine badge class
      let badgeClass = 'badge-needs-improvement';
      if (counselor.averageScore >= 90) {
        badgeClass = 'badge-excellent';
      } else if (counselor.averageScore >= 75) {
        badgeClass = 'badge-good';
      }

      // Determine trend icon
      let trendIcon = '↔';
      let trendClass = 'trend-neutral';
      if (counselor.trend === 'improving') {
        trendIcon = '↑';
        trendClass = 'trend-up';
      } else if (counselor.trend === 'declining') {
        trendIcon = '↓';
        trendClass = 'trend-down';
      }
```

```
        // Format date
        const reviewDate = counselor.lastReviewDate ? new
Date(counselor.lastReviewDate).toLocaleDateString() : 'N/A';

        counselorCard.innerHTML = `
          <div class="counselor-info">
            <div class="counselor-name">${counselor.name}</div>
            <div class="counselor-meta">
              <div class="meta-item"><span class="score-badge
${badgeClass}"></span> ${counselor.count} reviews</div>
              <div class="meta-item">Last review: ${reviewDate}</div>
            </div>
          </div>
          <div class="counselor-score">
            ${counselor.averageScore}% <span class="trend-icon
${trendClass}">${trendIcon}</span>
          </div>
        `;

        counselorCard.addEventListener('click', function() {
          // Switch to counselor details tab and select this counselor

document.querySelector('.tab-item[data-tab="counselor-details"]').click();
          document.getElementById('counselorSelect').value = counselor.name;
          loadCounselorDetails(counselor.name);
        });

        counselorListElement.appendChild(counselorCard);
      });
    }

    // Load counselors for dropdowns
```

```
  function loadCounselors() {
    google.script.run
      .withSuccessHandler(function(result) {
        allCounselors = result;
        populateCounselorDropdowns();
      })
      .withFailureHandler(function(error) {
        showError("Failed to load counselors: " + error);
      })
      .getTeamMembers();
  }

  // Populate counselor dropdowns
  function populateCounselorDropdowns() {
    const counselorSelect = document.getElementById('counselorSelect');
    const filterCounselorSelect =
document.getElementById('filterCounselorSelect');

    // Clear options except the first one
    counselorSelect.innerHTML = '<option value="">-- Select Counselor
--</option>';
    filterCounselorSelect.innerHTML = '<option value="">All
Counselors</option>';

    // Add counselors from team stats if available
    if (teamStats && teamStats.counselors) {
      teamStats.counselors.forEach(function(counselor) {
        const option1 = document.createElement('option');
        option1.value = counselor.name;
        option1.textContent = counselor.name;
        counselorSelect.appendChild(option1);

        const option2 = document.createElement('option');
```

```
        option2.value = counselor.name;
        option2.textContent = counselor.name;
        filterCounselorSelect.appendChild(option2);
      });
    }

    // Set up event listener for counselor selection
    counselorSelect.addEventListener('change', function() {
      const selectedCounselor = counselorSelect.value;
      if (selectedCounselor) {
        loadCounselorDetails(selectedCounselor);
      } else {
        document.getElementById('counselorDetailsContent').style.display =
'none';
        document.getElementById('counselorDetailsLoading').style.display =
'block';
        document.getElementById('counselorDetailsLoading').textContent =
'Please select a counselor to view details.';
      }
    });

    // Set up event listeners for filters
    filterCounselorSelect.addEventListener('change', filterReviews);
    document.getElementById('scoreRangeSelect').addEventListener('change',
filterReviews);
    document.getElementById('sortSelect').addEventListener('change',
filterReviews);
    document.getElementById('dateRangeSelect').addEventListener('change',
function() {
      if (counselorSelect.value) {
        loadCounselorDetails(counselorSelect.value);
      }
    });
```

```
  }

  // Load counselor details
  function loadCounselorDetails(counselorName) {
    document.getElementById('counselorDetailsLoading').style.display =
'block';
    document.getElementById('counselorDetailsContent').style.display =
'none';
    document.getElementById('counselorDetailsLoading').textContent = 'Loading
counselor details...';

    // Find counselor email based on name
    const counselorEmail = getEmailFromName(counselorName);

    if (!counselorEmail) {
      document.getElementById('counselorDetailsLoading').textContent =
'Counselor not found.';
      return;
    }

    google.script.run
      .withSuccessHandler(function(result) {
        renderCounselorDetails(counselorName, result);

        document.getElementById('counselorDetailsLoading').style.display =
'none';
        document.getElementById('counselorDetailsContent').style.display =
'block';
      })
      .withFailureHandler(function(error) {
        showError("Failed to load counselor details: " + error);
        document.getElementById('counselorDetailsLoading').style.display =
'none';
```

```
      })
      .getQualityStatsForCounselor(counselorEmail);
  }

  // Render counselor details
  function renderCounselorDetails(counselorName, data) {
    // Update counselor name
    document.getElementById('selectedCounselorName').textContent =
counselorName;

    // Update summary metrics
    document.getElementById('counselorAverageScore').textContent =
data.averageScore + '%';
    document.getElementById('counselorReviewCount').textContent = data.count;
    document.getElementById('counselorLastScore').textContent =
data.lastScore ? data.lastScore + '%' : 'N/A';

    // Update trend
    let trendDisplay = '↔';
    if (data.trend === 'improving') {
      trendDisplay = '↑';
    } else if (data.trend === 'declining') {
      trendDisplay = '↓';
    }
    document.getElementById('counselorTrend').innerHTML = trendDisplay;

    // Filter reviews for this counselor
    const counselorEmail = getEmailFromName(counselorName);

    // Get date range filter
    const dateRangeSelect = document.getElementById('dateRangeSelect');
    const daysFilter = parseInt(dateRangeSelect.value) || 0;
```

```
    // Filter by date if needed
    let filteredReviews = allReviews.filter(review =>
      review.counselor === counselorName ||
      (counselorEmail &&
review.counselor.toLowerCase().includes(counselorEmail.split('@')[0].toLowerCa
se()))
    );

    if (daysFilter > 0) {
      const cutoffDate = new Date();
      cutoffDate.setDate(cutoffDate.getDate() - daysFilter);

      filteredReviews = filteredReviews.filter(review => {
        const reviewDate = new Date(review.date);
        return reviewDate >= cutoffDate;
      });
    }

    // Sort by date (newest first)
    filteredReviews.sort((a, b) => new Date(b.date) - new Date(a.date));

    // Render review list
    const reviewHistoryList = document.getElementById('reviewHistoryList');
    reviewHistoryList.innerHTML = '';

    if (filteredReviews.length === 0) {
      reviewHistoryList.innerHTML = '<p>No reviews found for this counselor
in the selected time period.</p>';
      return;
    }

    filteredReviews.forEach(function(review) {
      const reviewCard = document.createElement('div');
```

```
        reviewCard.className = 'counselor-card';

        // Determine badge class
        let badgeClass = 'badge-needs-improvement';
        if (review.score >= 90) {
          badgeClass = 'badge-excellent';
        } else if (review.score >= 75) {
          badgeClass = 'badge-good';
        }

        // Format dates
        const reviewDate = new Date(review.date).toLocaleDateString();
        const evaluationDate = new
Date(review.reviewDate).toLocaleDateString();

        reviewCard.innerHTML = `
          <div class="counselor-info">
            <div class="counselor-name">Interaction ID:
${review.interactionId}</div>
            <div class="counselor-meta">
              <div class="meta-item">Date: ${reviewDate}</div>
              <div class="meta-item">Reviewed: ${evaluationDate}</div>
              <div class="meta-item">By: ${review.reviewer}</div>
            </div>
          </div>
          <div class="counselor-score">
            <span class="score-badge ${badgeClass}"></span> ${review.score}%
          </div>
        `;

        // Add event to expand review details
        reviewCard.addEventListener('click', function() {
          viewReviewDetails(review.id);
```

```
      });

      reviewHistoryList.appendChild(reviewCard);
    });
  }

  // Load all reviews
  function loadAllReviews() {
    document.getElementById('reviewsListLoading').style.display = 'block';
    document.getElementById('reviewsListContent').style.display = 'none';

    google.script.run
      .withSuccessHandler(function(result) {
        allReviews = result;
        filterReviews();

        document.getElementById('reviewsListLoading').style.display = 'none';
        document.getElementById('reviewsListContent').style.display =
'block';
      })
      .withFailureHandler(function(error) {
        showError("Failed to load reviews: " + error);
        document.getElementById('reviewsListLoading').style.display = 'none';
      })
      .getAllQualityReviews();
  }

  // Filter and sort reviews based on selected filters
  function filterReviews() {
    // Get filter values
    const counselorFilter =
document.getElementById('filterCounselorSelect').value;
```

```
    const scoreRangeFilter =
document.getElementById('scoreRangeSelect').value;
    const sortOption = document.getElementById('sortSelect').value;

    // Apply filters
    let filteredReviews = [...allReviews];

    // Filter by counselor
    if (counselorFilter) {
      filteredReviews = filteredReviews.filter(review => review.counselor ===
counselorFilter);
    }

    // Filter by score range
    if (scoreRangeFilter) {
      const [min, max] = scoreRangeFilter.split('-').map(Number);
      filteredReviews = filteredReviews.filter(review => {
        const score = parseInt(review.score);
        return score >= min && (max ? score <= max : true);
      });
    }

    // Apply sorting
    switch (sortOption) {
      case 'date-desc':
        filteredReviews.sort((a, b) => new Date(b.date) - new Date(a.date));
        break;
      case 'date-asc':
        filteredReviews.sort((a, b) => new Date(a.date) - new Date(b.date));
        break;
      case 'score-desc':
        filteredReviews.sort((a, b) => b.score - a.score);
        break;
```

```
      case 'score-asc':
        filteredReviews.sort((a, b) => a.score - b.score);
        break;
    }

    // Render filtered reviews
    renderReviewsList(filteredReviews);
  }

  // Render the reviews list
  function renderReviewsList(reviews) {
    const reviewsListElement = document.getElementById('reviewsList');
    reviewsListElement.innerHTML = '';

    if (reviews.length === 0) {
      reviewsListElement.innerHTML = '<p>No reviews match the selected
filters.</p>';
      return;
    }

    reviews.forEach(function(review) {
      const reviewCard = document.createElement('div');
      reviewCard.className = 'counselor-card';

      // Determine badge class
      let badgeClass = 'badge-needs-improvement';
      if (review.score >= 90) {
        badgeClass = 'badge-excellent';
      } else if (review.score >= 75) {
        badgeClass = 'badge-good';
      }

      // Format dates
```

```
      const reviewDate = new Date(review.date).toLocaleDateString();
      const evaluationDate = review.reviewDate ? new
Date(review.reviewDate).toLocaleDateString() : 'N/A';

      reviewCard.innerHTML = `
        <div class="counselor-info">
          <div class="counselor-name">${review.counselor}</div>
          <div class="counselor-meta">
            <div class="meta-item">ID: ${review.interactionId}</div>
            <div class="meta-item">Date: ${reviewDate}</div>
            <div class="meta-item">Reviewed: ${evaluationDate}</div>
          </div>
        </div>
        <div class="counselor-score">
          <span class="score-badge ${badgeClass}"></span> ${review.score}%
        </div>
      `;

      // Add event to expand review details
      reviewCard.addEventListener('click', function() {
        viewReviewDetails(review.id);
      });

      reviewsListElement.appendChild(reviewCard);
    });
  }

  // View review details
  function viewReviewDetails(reviewId) {
    // This would be implemented to show a modal with review details
    // For now, just show an alert
```

```
    alert('Viewing details for review ID: ' + reviewId + '\n\nThis functionality would show a detailed breakdown of the quality review scores and comments.');

    // In a real implementation, would call:
    // google.script.run
    //   .withSuccessHandler(function(result) {
    //     // Show modal with detailed review data
    //   })
    //   .getQualityReviewDetails(reviewId);
  }

  // Helper function to get email from name
  function getEmailFromName(name) {
    // This is a simple implementation - in a real system would use a more robust method
    const counselor = allCounselors.find(c => c.name === name);
    return counselor ? counselor.email : null;
  }

  // Show error message
  function showError(message) {
    const errorElement = document.getElementById('errorMessage');
    errorElement.textContent = message;
    errorElement.style.display = 'block';

    // Hide after 5 seconds
    setTimeout(function() {
      errorElement.style.display = 'none';
    }, 5000);
  }

  // Initialize on page load
```

```
   google.script.onload = function() {
     initialize();
   };
 </script>
</body>
</html>

   <div id="errorMessage" class="error-message">
     An error occurred while loading quality data.
   </div>

   <div class="tabs">
     <div class="tab-item active" data-tab="team-overview">Team Overview</div>
     <div class="tab-item" data-tab="counselor-details">Counselor
Details</div>
     <div class="tab-item" data-tab="reviews-list">Review List</div>
   </div>

   <!-- Team Overview Tab -->
   <div id="team-overview" class="tab-content active">
     <div id="teamOverviewLoading" class="loading">
       Loading team overview...
     </div>

     <div id="teamOverviewContent" style="display: none;">
       <div class="card">
         <h3>Team Performance</h3>

         <div class="team-summary">
           <div class="summary-metric">
             <div class="metric-value" id="teamAverageScore">--</div>
             <div class="metric-label">Average Score</div>
           </div>
```

```
        <div class="summary-metric">
          <div class="metric-value" id="totalReviews">--</div>
          <div class="metric-label">Total Reviews</div>
        </div>

        <div class="summary-metric">
          <div class="metric-value" id="counselorsReviewed">--</div>
          <div class="metric-label">Counselors Reviewed</div>
        </div>
      </div>

      <h4>Counselor Summary</h4>

      <div id="counselorList">
        <!-- Counselor cards will be added here -->
      </div>
    </div>
  </div>
</div>

<!-- Counselor Details Tab -->
<div id="counselor-details" class="tab-content">
  <div class="filters">
    <div class="filter-group">
      <label class="filter-label">Select Counselor</label>
      <select id="counselorSelect" class="filter-select">
        <option value="">-- Select Counselor --</option>
        <!-- Counselors will be populated here -->
      </select>
    </div>

    <div class="filter-group">
```

```
        <label class="filter-label">Date Range</label>
        <select id="dateRangeSelect" class="filter-select">
          <option value="30">Last 30 Days</option>
          <option value="90">Last 90 Days</option>
          <option value="180">Last 180 Days</option>
          <option value="365">Last Year</option>
          <option value="all">All Time</option>
        </select>
      </div>
    </div>

    <div id="counselorDetailsLoading" class="loading">
      Please select a counselor to view details.
    </div>

    <div id="counselorDetailsContent" style="display: none;">
      <div class="card">
        <h3 id="selectedCounselorName">Counselor Name</h3>

        <div class="team-summary">
          <div class="summary-metric">
            <div class="metric-value" id="counselorAverageScore">--</div>
            <div class="metric-label">Average Score</div>
          </div>

          <div class="summary-metric">
            <div class="metric-value" id="counselorReviewCount">--</div>
            <div class="metric-label">Reviews</div>
          </div>

          <div class="summary-metric">
            <div class="metric-value" id="counselorLastScore">--</div>
            <div class="metric-label">Last Score</div>
```

```
          </div>

          <div class="summary-metric">
            <div class="metric-value" id="counselorTrend">--</div>
            <div class="metric-label">Trend</div>
          </div>
        </div>

        <h4>Review History</h4>

        <div id="reviewHistoryList">
          <!-- Review history items will be added here -->
        </div>
      </div>
    </div>
  </div>
<!DOCTYPE html>
<html>
<head>
 <base target="_top">
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <link
href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500;700&fam
ily=Poppins:wght@500;600;700&display=swap" rel="stylesheet">
 <style>
   /* Variables for Trevor Project color scheme */
   :root {
     /* Primary Colors */
     --trevor-orange: #FF786E;
     --deep-blue: #001A4E;
     --purple: #9A3499;
     --teal: #137F6A;
```

```
    /* Secondary Colors */
    --light-blue: #4F52DE;
    --soft-yellow: #FFAD8D;
    --lavender: #B3AE4A;
    --light-purple: #F54AC;

    /* Tertiary Colors */
    --soft-green: #BAE2CE;
    --pale-yellow: #FFF2DF;
    --light-pink: #FBCBBE;
    --soft-lavender: #D1CFCC;

    /* Gradients */
    --primary-gradient: linear-gradient(135deg, var(--trevor-orange),
var(--light-pink));
    --calm-gradient: linear-gradient(135deg, var(--light-blue),
var(--soft-green));
    --support-gradient: linear-gradient(45deg, var(--purple),
var(--light-purple));
    --background-gradient: linear-gradient(180deg, var(--pale-yellow),
#FFFFFF);

    /* Spacing */
    --spacing-xs: 4px;
    --spacing-sm: 8px;
    --spacing-md: 16px;
    --spacing-lg: 24px;
    --spacing-xl: 32px;

    /* Borders */
    --border-radius-sm: 8px;
    --border-radius-md: 12px;
```

```
  --border-radius-lg: 16px;
}

body {
  font-family: 'Roboto', sans-serif;
  margin: 0;
  padding: 0;
  background-color: var(--pale-yellow);
  color: var(--deep-blue);
}

.container {
  max-width: 900px;
  margin: 0 auto;
  padding: var(--spacing-md);
}

h1, h2, h3, h4 {
  font-family: 'Poppins', sans-serif;
  color: var(--deep-blue);
}

.header {
  margin-bottom: var(--spacing-lg);
  text-align: center;
}

.card {
  background: white;
  border-radius: var(--border-radius-md);
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.05);
  padding: var(--spacing-lg);
  margin-bottom: var(--spacing-lg);
```

```
  position: relative;
}

.card::before {
  content: '';
  position: absolute;
  left: 0;
  top: 0;
  height: 100%;
  width: 3px;
  background: var(--light-blue);
}

.form-group {
  margin-bottom: var(--spacing-md);
}

label {
  display: block;
  font-weight: 500;
  margin-bottom: var(--spacing-xs);
}

input, select, textarea {
  width: 100%;
  padding: var(--spacing-sm);
  border: 1px solid #ddd;
  border-radius: var(--border-radius-sm);
  font-family: 'Roboto', sans-serif;
  font-size: 16px;
}

textarea {
```

```
  resize: vertical;
  min-height: 80px;
}

.btn {
  border: none;
  padding: var(--spacing-md) var(--spacing-lg);
  border-radius: var(--border-radius-sm);
  font-weight: 500;
  cursor: pointer;
  transition: all 0.2s ease;
}

.btn-primary {
  background: var(--primary-gradient);
  color: white;
}

.btn-primary:hover {
  box-shadow: 0 4px 8px rgba(255, 120, 110, 0.3);
  transform: translateY(-2px);
}

.btn-secondary {
  background: white;
  color: var(--deep-blue);
  border: 1px solid #ddd;
}

.btn-secondary:hover {
  background: #f5f5f5;
}
```

```css
.form-actions {
  display: flex;
  justify-content: space-between;
  margin-top: var(--spacing-lg);
}

.counselor-selector {
  padding: var(--spacing-md);
  margin-bottom: var(--spacing-lg);
  background: white;
  border-radius: var(--border-radius-md);
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.05);
}

.score-radio-group {
  display: flex;
  flex-direction: row;
  justify-content: space-between;
  margin: var(--spacing-xs) 0;
}

.score-radio {
  display: flex;
  flex-direction: column;
  align-items: center;
  margin-right: var(--spacing-md);
}

.score-radio label {
  margin-top: var(--spacing-xs);
  font-size: 12px;
  text-align: center;
}
```

```css
.criteria-section {
  margin-bottom: var(--spacing-lg);
}

.criteria-item {
  margin-bottom: var(--spacing-md);
  padding-bottom: var(--spacing-md);
  border-bottom: 1px solid #eee;
}

.criteria-header {
  margin-bottom: var(--spacing-sm);
  font-weight: 500;
}

.criteria-note {
  margin-top: var(--spacing-sm);
}

.criteria-note-input {
  margin-top: var(--spacing-xs);
}

.success-message {
  background-color: var(--soft-green);
  border-radius: var(--border-radius-sm);
  padding: var(--spacing-md);
  margin-bottom: var(--spacing-md);
  display: none;
}

.error-message {
```

```
  background-color: var(--light-pink);
  border-radius: var(--border-radius-sm);
  padding: var(--spacing-md);
  margin-bottom: var(--spacing-md);
  display: none;
}

.results-card {
  background: white;
  border-radius: var(--border-radius-md);
  padding: var(--spacing-lg);
  margin-bottom: var(--spacing-lg);
  text-align: center;
  display: none;
}

.score-result {
  font-size: 48px;
  font-weight: 700;
  color: var(--deep-blue);
  margin-bottom: var(--spacing-md);
}

.score-label {
  font-size: 18px;
  margin-bottom: var(--spacing-sm);
}

.score-status {
  display: inline-block;
  padding: var(--spacing-sm) var(--spacing-md);
  border-radius: var(--border-radius-sm);
  font-weight: 500;
```

```
    margin: var(--spacing-md) 0;
  }

  .status-excellent {
    background-color: var(--soft-green);
    color: var(--deep-blue);
  }

  .status-good {
    background-color: var(--soft-yellow);
    color: var(--deep-blue);
  }

  .status-needs-improvement {
    background-color: var(--light-pink);
    color: var(--deep-blue);
  }
 </style>
</head>
<body>
 <div class="container">
  <div class="header">
    <h1>Quality Review</h1>
    <p>Evaluate counselor interactions against quality standards</p>
  </div>

  <div id="successMessage" class="success-message">
    Quality review saved successfully!
  </div>

  <div id="errorMessage" class="error-message">
    An error occurred while saving the quality review. Please try again.
  </div>
```

```
   <div id="resultsCard" class="results-card">
     <h2>Quality Review Results</h2>
     <div class="score-label">Overall Score</div>
     <div class="score-result" id="finalScore">--</div>
     <div class="score-status" id="scoreStatus">--</div>
     <p>Thank you for completing this quality review. Your feedback helps
improve our service quality.</p>
     <button type="button" class="btn btn-primary"
onclick="startNewReview()">Start New Review</button>
   </div>

   <div id="reviewForm">
     <div class="counselor-selector card">
       <div class="form-group">
         <label for="counselorSelect">Select Counselor</label>
         <select id="counselorSelect" onchange="counselorSelected()">
           <option value="">-- Select Counselor --</option>
           <!-- Counselors will be populated here -->
         </select>
       </div>

       <div id="counselorDetails" style="display: none;">
         <div class="form-group">
           <label for="interactionDate">Interaction Date</label>
           <input type="date" id="interactionDate" name="interactionDate"
required>
         </div>

         <div class="form-group">
           <label for="interactionId">Interaction ID</label>
           <input type="text" id="interactionId" name="interactionId"
placeholder="Enter the interaction ID or reference number" required>
```

```
        </div>
      </div>
    </div>

    <div id="criteriaForm" style="display: none;" class="card">
      <h3>Quality Evaluation Criteria</h3>
      <p>Evaluate each area using the point scale: 2 points (Fully Met), 1 point (Partially Met), 0 points (Not Met), or N/A if not applicable.</p>

      <div class="criteria-section">
        <h4>Call Opening</h4>

        <div class="criteria-item">
          <div class="criteria-header">Answered interaction in less than 20 seconds of joining</div>
          <div class="score-radio-group">
            <div class="score-radio">
              <input type="radio" id="opening1_2" name="opening1" value="2">
              <label for="opening1_2">Fully Met</label>
            </div>
            <div class="score-radio">
              <input type="radio" id="opening1_1" name="opening1" value="1">
              <label for="opening1_1">Partially Met</label>
            </div>
            <div class="score-radio">
              <input type="radio" id="opening1_0" name="opening1" value="0">
              <label for="opening1_0">Not Met</label>
            </div>
            <div class="score-radio">
              <input type="radio" id="opening1_na" name="opening1" value="NA">
              <label for="opening1_na">N/A</label>
            </div>
```

```
            </div>
            <div class="criteria-note">
              <label for="opening1_note">Notes:</label>
              <textarea id="opening1_note"
class="criteria-note-input"></textarea>
            </div>
          </div>

          <div class="criteria-item">
            <div class="criteria-header">Opening message completed with
tact</div>
            <div class="score-radio-group">
              <div class="score-radio">
                <input type="radio" id="opening2_2" name="opening2" value="2">
                <label for="opening2_2">Fully Met</label>
              </div>
              <div class="score-radio">
                <input type="radio" id="opening2_1" name="opening2" value="1">
                <label for="opening2_1">Partially Met</label>
              </div>
              <div class="score-radio">
                <input type="radio" id="opening2_0" name="opening2" value="0">
                <label for="opening2_0">Not Met</label>
              </div>
              <div class="score-radio">
                <input type="radio" id="opening2_na" name="opening2"
value="NA">
                <label for="opening2_na">N/A</label>
              </div>
            </div>
            <div class="criteria-note">
              <label for="opening2_note">Notes:</label>
```

```
            <textarea id="opening2_note"
class="criteria-note-input"></textarea>
          </div>
        </div>
      </div>

      <div class="criteria-section">
        <h4>Risk Assessment</h4>

        <div class="criteria-item">
          <div class="criteria-header">SASS/Screener questions attempted at
least twice (unless contact requested otherwise)</div>
          <div class="score-radio-group">
            <div class="score-radio">
              <input type="radio" id="risk1_2" name="risk1" value="2">
              <label for="risk1_2">Fully Met</label>
            </div>
            <div class="score-radio">
              <input type="radio" id="risk1_1" name="risk1" value="1">
              <label for="risk1_1">Partially Met</label>
            </div>
            <div class="score-radio">
              <input type="radio" id="risk1_0" name="risk1" value="0">
              <label for="risk1_0">Not Met</label>
            </div>
            <div class="score-radio">
              <input type="radio" id="risk1_na" name="risk1" value="NA">
              <label for="risk1_na">N/A</label>
            </div>
          </div>
          <div class="criteria-note">
            <label for="risk1_note">Notes:</label>
            <textarea id="risk1_note" class="criteria-note-input"></textarea>
```

```
          </div>
        </div>

        <div class="criteria-item">
          <div class="criteria-header">Used creative non-judgmental empathy
and built rapport</div>
          <div class="score-radio-group">
            <div class="score-radio">
              <input type="radio" id="risk2_2" name="risk2" value="2">
              <label for="risk2_2">Fully Met</label>
            </div>
            <div class="score-radio">
              <input type="radio" id="risk2_1" name="risk2" value="1">
              <label for="risk2_1">Partially Met</label>
            </div>
            <div class="score-radio">
              <input type="radio" id="risk2_0" name="risk2" value="0">
              <label for="risk2_0">Not Met</label>
            </div>
            <div class="score-radio">
              <input type="radio" id="risk2_na" name="risk2" value="NA">
              <label for="risk2_na">N/A</label>
            </div>
          </div>
          <div class="criteria-note">
            <label for="risk2_note">Notes:</label>
            <textarea id="risk2_note" class="criteria-note-input"></textarea>
          </div>
        </div>
      </div>

      <div class="criteria-section">
        <h4>Communication Skills</h4>
```

```
        <div class="criteria-item">
          <div class="criteria-header">Demonstrated active listening through
reflective responses</div>
          <div class="score-radio-group">
            <div class="score-radio">
              <input type="radio" id="comm1_2" name="comm1" value="2">
              <label for="comm1_2">Fully Met</label>
            </div>
            <div class="score-radio">
              <input type="radio" id="comm1_1" name="comm1" value="1">
              <label for="comm1_1">Partially Met</label>
            </div>
            <div class="score-radio">
              <input type="radio" id="comm1_0" name="comm1" value="0">
              <label for="comm1_0">Not Met</label>
            </div>
            <div class="score-radio">
              <input type="radio" id="comm1_na" name="comm1" value="NA">
              <label for="comm1_na">N/A</label>
            </div>
          </div>
          <div class="criteria-note">
            <label for="comm1_note">Notes:</label>
            <textarea id="comm1_note" class="criteria-note-input"></textarea>
          </div>
        </div>

        <div class="criteria-item">
          <div class="criteria-header">Used appropriate LGBTQ+ inclusive
language</div>
          <div class="score-radio-group">
            <div class="score-radio">
```

```
              <input type="radio" id="comm2_2" name="comm2" value="2">
              <label for="comm2_2">Fully Met</label>
            </div>
            <div class="score-radio">
              <input type="radio" id="comm2_1" name="comm2" value="1">
              <label for="comm2_1">Partially Met</label>
            </div>
            <div class="score-radio">
              <input type="radio" id="comm2_0" name="comm2" value="0">
              <label for="comm2_0">Not Met</label>
            </div>
            <div class="score-radio">
              <input type="radio" id="comm2_na" name="comm2" value="NA">
              <label for="comm2_na">N/A</label>
            </div>
          </div>
          <div class="criteria-note">
            <label for="comm2_note">Notes:</label>
            <textarea id="comm2_note" class="criteria-note-input"></textarea>
          </div>
        </div>

        <div class="criteria-item">
          <div class="criteria-header">Responded to emotional content with
appropriate empathy</div>
          <div class="score-radio-group">
            <div class="score-radio">
              <input type="radio" id="comm3_2" name="comm3" value="2">
              <label for="comm3_2">Fully Met</label>
            </div>
            <div class="score-radio">
              <input type="radio" id="comm3_1" name="comm3" value="1">
              <label for="comm3_1">Partially Met</label>
```

```
            </div>
            <div class="score-radio">
              <input type="radio" id="comm3_0" name="comm3" value="0">
              <label for="comm3_0">Not Met</label>
            </div>
            <div class="score-radio">
              <input type="radio" id="comm3_na" name="comm3" value="NA">
              <label for="comm3_na">N/A</label>
            </div>
          </div>
          <div class="criteria-note">
            <label for="comm3_note">Notes:</label>
            <textarea id="comm3_note" class="criteria-note-input"></textarea>
          </div>
        </div>
      </div>

      <div class="criteria-section">
        <h4>Support Effectiveness</h4>

        <div class="criteria-item">
          <div class="criteria-header">Provided appropriate resources
relevant to the contact's needs</div>
          <div class="score-radio-group">
            <div class="score-radio">
              <input type="radio" id="support1_2" name="support1" value="2">
              <label for="support1_2">Fully Met</label>
            </div>
            <div class="score-radio">
              <input type="radio" id="support1_1" name="support1" value="1">
              <label for="support1_1">Partially Met</label>
            </div>
            <div class="score-radio">
```

```
              <input type="radio" id="support1_0" name="support1" value="0">
              <label for="support1_0">Not Met</label>
            </div>
            <div class="score-radio">
              <input type="radio" id="support1_na" name="support1"
value="NA">
              <label for="support1_na">N/A</label>
            </div>
          </div>
          <div class="criteria-note">
            <label for="support1_note">Notes:</label>
            <textarea id="support1_note"
class="criteria-note-input"></textarea>
          </div>
        </div>

        <div class="criteria-item">
          <div class="criteria-header">Helped identify coping strategies or
action steps</div>
          <div class="score-radio-group">
            <div class="score-radio">
              <input type="radio" id="support2_2" name="support2" value="2">
              <label for="support2_2">Fully Met</label>
            </div>
            <div class="score-radio">
              <input type="radio" id="support2_1" name="support2" value="1">
              <label for="support2_1">Partially Met</label>
            </div>
            <div class="score-radio">
              <input type="radio" id="support2_0" name="support2" value="0">
              <label for="support2_0">Not Met</label>
            </div>
            <div class="score-radio">
```

```
              <input type="radio" id="support2_na" name="support2"
value="NA">
              <label for="support2_na">N/A</label>
            </div>
          </div>
          <div class="criteria-note">
            <label for="support2_note">Notes:</label>
            <textarea id="support2_note"
class="criteria-note-input"></textarea>
          </div>
        </div>
      </div>

      <div class="criteria-section">
        <h4>Call Closure</h4>

        <div class="criteria-item">
          <div class="criteria-header">Summarized key points of the
interaction</div>
          <div class="score-radio-group">
            <div class="score-radio">
              <input type="radio" id="closure1_2" name="closure1" value="2">
              <label for="closure1_2">Fully Met</label>
            </div>
            <div class="score-radio">
              <input type="radio" id="closure1_1" name="closure1" value="1">
              <label for="closure1_1">Partially Met</label>
            </div>
            <div class="score-radio">
              <input type="radio" id="closure1_0" name="closure1" value="0">
              <label for="closure1_0">Not Met</label>
            </div>
            <div class="score-radio">
```

```
              <input type="radio" id="closure1_na" name="closure1"
value="NA">
              <label for="closure1_na">N/A</label>
            </div>
          </div>
          <div class="criteria-note">
            <label for="closure1_note">Notes:</label>
            <textarea id="closure1_note"
class="criteria-note-input"></textarea>
          </div>
        </div>

        <div class="criteria-item">
          <div class="criteria-header">Completed closure process
appropriately</div>
          <div class="score-radio-group">
            <div class="score-radio">
              <input type="radio" id="closure2_2" name="closure2" value="2">
              <label for="closure2_2">Fully Met</label>
            </div>
            <div class="score-radio">
              <input type="radio" id="closure2_1" name="closure2" value="1">
              <label for="closure2_1">Partially Met</label>
            </div>
            <div class="score-radio">
              <input type="radio" id="closure2_0" name="closure2" value="0">
              <label for="closure2_0">Not Met</label>
            </div>
            <div class="score-radio">
              <input type="radio" id="closure2_na" name="closure2"
value="NA">
              <label for="closure2_na">N/A</label>
            </div>
```

```
            </div>
            <div class="criteria-note">
              <label for="closure2_note">Notes:</label>
              <textarea id="closure2_note"
class="criteria-note-input"></textarea>
            </div>
          </div>
        </div>

        <div class="form-group">
          <label for="overallFeedback">Overall Feedback</label>
          <textarea id="overallFeedback" rows="4" placeholder="Provide overall
feedback about the interaction"></textarea>
        </div>

        <div class="form-actions">
          <button type="button" class="btn btn-secondary"
onclick="cancelReview()">Cancel</button>
          <button type="button" class="btn btn-primary"
onclick="submitReview()">Submit Review</button>
        </div>
      </div>
    </div>
  </div>
  <script>
    // Global variables
    let counselors = [];
    let selectedCounselor = null;

    // Initialize the form
    function initialize() {
      // Get counselors
      google.script.run
```

```
      .withSuccessHandler(function(result) {
        counselors = result;
        populateCounselors();
      })
      .withFailureHandler(function(error) {
        showError("Failed to load counselors: " + error);
      })
      .getTeamMembers();
}

// Populate counselors dropdown
function populateCounselors() {
  const select = document.getElementById('counselorSelect');
  select.innerHTML = '<option value="">-- Select Counselor --</option>';

  counselors.forEach(function(counselor) {
    const option = document.createElement('option');
    option.value = counselor.email;
    option.textContent = counselor.name;
    select.appendChild(option);
  });
}

// Handle counselor selection
function counselorSelected() {
  const select = document.getElementById('counselorSelect');
  const email = select.value;

  if (!email) {
    document.getElementById('counselorDetails').style.display = 'none';
    document.getElementById('criteriaForm').style.display = 'none';
    return;
  }
```

```
    selectedCounselor = counselors.find(c => c.email === email);
    document.getElementById('counselorDetails').style.display = 'block';
    document.getElementById('criteriaForm').style.display = 'block';

    // Set default interaction date to today
    const today = new Date().toISOString().split('T')[0];
    document.getElementById('interactionDate').value = today;
  }

  // Cancel review
  function cancelReview() {
    // Reset form
    document.getElementById('counselorSelect').value = '';
    document.getElementById('counselorDetails').style.display = 'none';
    document.getElementById('criteriaForm').style.display = 'none';
    document.getElementById('interactionDate').value = '';
    document.getElementById('interactionId').value = '';
    document.getElementById('overallFeedback').value = '';

    // Reset radio buttons
    const radioButtons = document.querySelectorAll('input[type="radio"]');
    radioButtons.forEach(function(radio) {
      radio.checked = false;
    });

    // Reset text areas
    const textAreas =
document.querySelectorAll('textarea.criteria-note-input');
    textAreas.forEach(function(textarea) {
      textarea.value = '';
    });
```

```
    // Hide messages
    document.getElementById('successMessage').style.display = 'none';
    document.getElementById('errorMessage').style.display = 'none';
}

// Submit review
function submitReview() {
    // Validate required fields
    const counselorSelect = document.getElementById('counselorSelect');
    const interactionDate = document.getElementById('interactionDate');
    const interactionId = document.getElementById('interactionId');

    if (!counselorSelect.value) {
        showError("Please select a counselor");
        return;
    }

    if (!interactionDate.value) {
        showError("Please enter the interaction date");
        return;
    }

    if (!interactionId.value) {
        showError("Please enter the interaction ID");
        return;
    }

    // Collect scores
    const scoreCategories = [
        'opening1', 'opening2',
        'risk1', 'risk2',
        'comm1', 'comm2', 'comm3',
        'support1', 'support2',
```

```
      'closure1', 'closure2'
    ];

    const scores = [];
    let valid = true;

    scoreCategories.forEach(function(category) {
      const selectedRadio =
document.querySelector(`input[name="${category}"]:checked`);

      if (!selectedRadio) {
        showError(`Please rate all criteria (missing: ${category})`);
        valid = false;
        return;
      }

      const noteId = `${category}_note`;
      const note = document.getElementById(noteId).value;

      scores.push({
        category: category,
        points: selectedRadio.value === 'NA' ? 'NA' :
parseInt(selectedRadio.value),
        note: note
      });
    });

    if (!valid) {
      return;
    }

    // Create review data object
    const reviewData = {
```

```
      counselorName: selectedCounselor.name,
      counselorEmail: selectedCounselor.email,
      date: interactionDate.value,
      interactionId: interactionId.value,
      scores: scores,
      overallFeedback: document.getElementById('overallFeedback').value
    };

    // Submit to server
    google.script.run
      .withSuccessHandler(function(result) {
        if (result.success) {
          // Show success and results
          document.getElementById('successMessage').style.display = 'block';
          document.getElementById('errorMessage').style.display = 'none';

          // Show results card
          document.getElementById('reviewForm').style.display = 'none';
          document.getElementById('resultsCard').style.display = 'block';

          // Update score
          document.getElementById('finalScore').textContent =
result.totalScore + '%';

          // Update status label
          const statusElement = document.getElementById('scoreStatus');
          let statusClass = '';
          let statusText = '';

          if (result.totalScore >= 90) {
            statusClass = 'status-excellent';
            statusText = 'Excellent';
          } else if (result.totalScore >= 75) {
```

```
            statusClass = 'status-good';
            statusText = 'Good';
          } else {
            statusClass = 'status-needs-improvement';
            statusText = 'Needs Improvement';
          }

          statusElement.className = 'score-status ' + statusClass;
          statusElement.textContent = statusText;
        } else {
          showError(result.message);
        }
      })
      .withFailureHandler(function(error) {
        showError("Failed to save review: " + error);
      })
      .saveQualityReview(reviewData);
}

// Start a new review
function startNewReview() {
  // Reset form and show it again
  cancelReview();
  document.getElementById('reviewForm').style.display = 'block';
  document.getElementById('resultsCard').style.display = 'none';
}

// Show error message
function showError(message) {
  const errorElement = document.getElementById('errorMessage');
  errorElement.textContent = message;
  errorElement.style.display = 'block';
  document.getElementById('successMessage').style.display = 'none';
```

```
    // Scroll to error
    errorElement.scrollIntoView({ behavior: 'smooth' });
  }

  // Initialize on page load
  google.script.onload = function() {
    initialize();
  };
 </script>
</body>
</html>
<!DOCTYPE html>
<html>
<head>
 <base target="_top">
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <link
href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500;700&fam
ily=Poppins:wght@500;600;700&display=swap" rel="stylesheet">
 <style>
  /* Variables for 988 Lifeline LGBTQIA+ Services color scheme */
  :root {
    /* Primary Colors */
    --trevor-orange: #FF786E;
    --deep-blue: #001A4E;
    --purple: #9A3499;
    --teal: #137F6A;

    /* Secondary Colors */
    --light-blue: #4F52DE;
    --soft-yellow: #FFAD8D;
```

```
    --lavender: #B3AE4A;
    --light-purple: #F54AC;

    /* Tertiary Colors */
    --soft-green: #BAE2CE;
    --pale-yellow: #FFF2DF;
    --light-pink: #FBCBBE;
    --soft-lavender: #D1CFCC;

    /* Gradients */
    --primary-gradient: linear-gradient(135deg, var(--trevor-orange),
var(--light-pink));
    --calm-gradient: linear-gradient(135deg, var(--light-blue),
var(--soft-green));
    --support-gradient: linear-gradient(45deg, var(--purple),
var(--light-purple));
    --background-gradient: linear-gradient(180deg, var(--pale-yellow),
#FFFFFF);

    /* Spacing */
    --spacing-xs: 4px;
    --spacing-sm: 8px;
    --spacing-md: 16px;
    --spacing-lg: 24px;
    --spacing-xl: 32px;

    /* Borders */
    --border-radius-sm: 8px;
    --border-radius-md: 12px;
    --border-radius-lg: 16px;
  }

  body {
```

```
  font-family: 'Roboto', sans-serif;
  margin: 0;
  padding: 0;
  background-color: var(--pale-yellow);
  color: var(--deep-blue);
}

.container {
  max-width: 900px;
  margin: 0 auto;
  padding: var(--spacing-md);
}

.header {
  margin-bottom: var(--spacing-lg);
  text-align: center;
}

h1, h2, h3, h4 {
  font-family: 'Poppins', sans-serif;
  color: var(--deep-blue);
}

.card {
  background: white;
  border-radius: var(--border-radius-md);
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.05);
  padding: var(--spacing-lg);
  margin-bottom: var(--spacing-lg);
  position: relative;
}

.card::before {
```

```
  content: '';
  position: absolute;
  left: 0;
  top: 0;
  height: 100%;
  width: 3px;
  background: var(--purple);
}

.tabs {
  display: flex;
  border-bottom: 1px solid #ddd;
  margin-bottom: var(--spacing-lg);
}

.tab {
  padding: var(--spacing-sm) var(--spacing-lg);
  cursor: pointer;
  border-bottom: 2px solid transparent;
  transition: all 0.2s ease;
}

.tab.active {
  border-bottom-color: var(--trevor-orange);
  font-weight: 500;
}

.tab-content {
  display: none;
}

.tab-content.active {
  display: block;
```

```css
}

.schedule-controls {
  display: flex;
  justify-content: space-between;
  align-items: center;
  margin-bottom: var(--spacing-md);
}

.date-navigation {
  display: flex;
  align-items: center;
  gap: var(--spacing-sm);
}

.nav-button {
  background: none;
  border: none;
  font-size: 24px;
  cursor: pointer;
  color: var(--deep-blue);
  padding: var(--spacing-xs);
}

.current-week {
  font-weight: 500;
  min-width: 200px;
  text-align: center;
}

.schedule-grid {
  display: grid;
  grid-template-columns: 120px repeat(7, 1fr);
```

```
  gap: 1px;
  background-color: #eee;
  border: 1px solid #eee;
  border-radius: var(--border-radius-sm);
  overflow: hidden;
}

.grid-header {
  background-color: var(--deep-blue);
  color: white;
  padding: var(--spacing-sm);
  text-align: center;
  font-weight: 500;
}

.grid-shift {
  background-color: var(--deep-blue);
  color: white;
  padding: var(--spacing-sm);
  font-weight: 500;
}

.grid-cell {
  background-color: white;
  padding: var(--spacing-sm);
  min-height: 60px;
  position: relative;
}

.grid-cell.weekend {
  background-color: #f9f9f9;
}
```

```css
.grid-cell.today {
  background-color: var(--pale-yellow);
}

.counselor-slot {
  padding: var(--spacing-xs) var(--spacing-sm);
  margin-bottom: var(--spacing-xs);
  border-radius: var(--border-radius-sm);
  font-size: 14px;
  background-color: var(--soft-lavender);
  cursor: pointer;
}

.add-counselor {
  position: absolute;
  bottom: var(--spacing-xs);
  right: var(--spacing-xs);
  width: 24px;
  height: 24px;
  border-radius: 50%;
  background-color: var(--light-blue);
  color: white;
  display: flex;
  align-items: center;
  justify-content: center;
  font-size: 18px;
  cursor: pointer;
}

.form-group {
  margin-bottom: var(--spacing-md);
}
```

```css
label {
  display: block;
  font-weight: 500;
  margin-bottom: var(--spacing-xs);
}

input, select, textarea {
  width: 100%;
  padding: var(--spacing-sm);
  border: 1px solid #ddd;
  border-radius: var(--border-radius-sm);
  font-family: 'Roboto', sans-serif;
}

.form-row {
  display: flex;
  gap: var(--spacing-md);
}

.form-col {
  flex: 1;
}

.btn {
  border: none;
  padding: var(--spacing-sm) var(--spacing-md);
  border-radius: var(--border-radius-sm);
  font-weight: 500;
  cursor: pointer;
  transition: all 0.2s ease;
}

.btn-primary {
```

```
  background: var(--primary-gradient);
  color: white;
}

.btn-primary:hover {
  box-shadow: 0 4px 8px rgba(255, 120, 110, 0.3);
  transform: translateY(-2px);
}

.btn-secondary {
  background: white;
  color: var(--deep-blue);
  border: 1px solid #ddd;
}

.btn-secondary:hover {
  background: #f5f5f5;
}

.action-buttons {
  display: flex;
  justify-content: flex-end;
  gap: var(--spacing-sm);
  margin-top: var(--spacing-md);
}

.success-message {
  background-color: var(--soft-green);
  color: var(--deep-blue);
  padding: var(--spacing-md);
  border-radius: var(--border-radius-sm);
  margin-bottom: var(--spacing-lg);
  text-align: center;
```

```
  display: none;
}

.error-message {
  background-color: var(--light-pink);
  color: var(--deep-blue);
  padding: var(--spacing-md);
  border-radius: var(--border-radius-sm);
  margin-bottom: var(--spacing-lg);
  text-align: center;
  display: none;
}

.modal {
  display: none;
  position: fixed;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  background-color: rgba(0, 0, 0, 0.5);
  z-index: 1000;
  justify-content: center;
  align-items: center;
}

.modal-content {
  background: white;
  border-radius: var(--border-radius-md);
  max-width: 500px;
  width: 100%;
  padding: var(--spacing-lg);
  position: relative;
```

```
}

.modal-close {
  position: absolute;
  top: var(--spacing-sm);
  right: var(--spacing-sm);
  font-size: 24px;
  cursor: pointer;
  background: none;
  border: none;
}

.list-view-item {
  padding: var(--spacing-sm);
  border-bottom: 1px solid #eee;
  display: flex;
  justify-content: space-between;
  align-items: center;
}

.list-view-item:last-child {
  border-bottom: none;
}

.counselor-info {
  flex: 1;
}

.counselor-name {
  font-weight: 500;
}

.counselor-date {
```

```
    font-size: 14px;
    color: #666;
  }

  .status-badge {
    padding: 2px 8px;
    border-radius: 12px;
    font-size: 12px;
    background-color: var(--soft-lavender);
  }

  .loading {
    text-align: center;
    padding: var(--spacing-lg);
    color: #666;
  }

  @media (max-width: 768px) {
    .schedule-grid {
      grid-template-columns: 100px repeat(7, 1fr);
      font-size: 14px;
    }

    .form-row {
      flex-direction: column;
      gap: 0;
    }
  }
 </style>
</head>
<body>
 <div class="container">
   <div class="header">
```

```
    <h1>Schedule Manager</h1>
    <p>Manage counselor schedules for the 988 Lifeline LGBTQIA+ Crisis Support Team</p>
  </div>

  <div id="successMessage" class="success-message">
    Schedule updated successfully!
  </div>

  <div id="errorMessage" class="error-message">
    An error occurred. Please try again.
  </div>

  <div class="tabs">
    <div class="tab active" data-tab="week-view">Week View</div>
    <div class="tab" data-tab="list-view">List View</div>
    <div class="tab" data-tab="add-shift">Add Shift</div>
  </div>

  <!-- Week View Tab -->
  <div id="week-view" class="tab-content active">
    <div class="schedule-controls">
      <div class="date-navigation">
        <button class="nav-button" onclick="previousWeek()">&lt;</button>
        <div class="current-week" id="currentWeekLabel">March 10 - March 16, 2025</div>
        <button class="nav-button" onclick="nextWeek()">&gt;</button>
      </div>

      <div>
        <button class="btn btn-secondary" onclick="goToToday()">Today</button>
      </div>
```

```
    </div>

    <div class="card">
      <div id="weekViewLoading" class="loading">
        Loading schedule...
      </div>

      <div id="scheduleGrid" class="schedule-grid" style="display: none;">
        <!-- Grid headers -->
        <div class="grid-header">Shift</div>
        <div class="grid-header">Sun</div>
        <div class="grid-header">Mon</div>
        <div class="grid-header">Tue</div>
        <div class="grid-header">Wed</div>
        <div class="grid-header">Thu</div>
        <div class="grid-header">Fri</div>
        <div class="grid-header">Sat</div>

        <!-- Morning Shift -->
        <div class="grid-shift">Morning<br>(8am-4pm)</div>
        <div id="sun-morning" class="grid-cell weekend"></div>
        <div id="mon-morning" class="grid-cell"></div>
        <div id="tue-morning" class="grid-cell"></div>
        <div id="wed-morning" class="grid-cell"></div>
        <div id="thu-morning" class="grid-cell"></div>
        <div id="fri-morning" class="grid-cell"></div>
        <div id="sat-morning" class="grid-cell weekend"></div>

        <!-- Afternoon Shift -->
        <div class="grid-shift">Afternoon<br>(4pm-12am)</div>
        <div id="sun-afternoon" class="grid-cell weekend"></div>
        <div id="mon-afternoon" class="grid-cell"></div>
        <div id="tue-afternoon" class="grid-cell"></div>
```

```
        <div id="wed-afternoon" class="grid-cell"></div>
        <div id="thu-afternoon" class="grid-cell"></div>
        <div id="fri-afternoon" class="grid-cell"></div>
        <div id="sat-afternoon" class="grid-cell weekend"></div>

        <!-- Night Shift -->
        <div class="grid-shift">Night<br>(12am-8am)</div>
        <div id="sun-night" class="grid-cell weekend"></div>
        <div id="mon-night" class="grid-cell"></div>
        <div id="tue-night" class="grid-cell"></div>
        <div id="wed-night" class="grid-cell"></div>
        <div id="thu-night" class="grid-cell"></div>
        <div id="fri-night" class="grid-cell"></div>
        <div id="sat-night" class="grid-cell weekend"></div>
      </div>
    </div>
  </div>

  <!-- List View Tab -->
  <div id="list-view" class="tab-content">
    <div class="form-row" style="margin-bottom: 20px;">
      <div class="form-col">
        <div class="form-group">
          <label for="listViewFilter">Filter By</label>
          <select id="listViewFilter" onchange="filterListView()">
            <option value="all">All Shifts</option>
            <option value="future">Upcoming Shifts</option>
            <option value="past">Past Shifts</option>
            <option value="counselor">By Counselor</option>
          </select>
        </div>
      </div>
```

```
      <div class="form-col" id="counselorFilterContainer" style="display:
none;">
        <div class="form-group">
          <label for="counselorFilter">Select Counselor</label>
          <select id="counselorFilter" onchange="filterListView()">
            <option value="">-- Select Counselor --</option>
            <!-- Counselors will be populated here -->
          </select>
        </div>
      </div>
    </div>

    <div class="card">
      <div id="listViewLoading" class="loading">
        Loading shifts...
      </div>

      <div id="shiftsList">
        <!-- Shifts will be listed here -->
      </div>
    </div>
  </div>

  <!-- Add Shift Tab -->
  <div id="add-shift" class="tab-content">
    <div class="card">
      <h3>Add New Shift</h3>

      <form id="addShiftForm">
        <div class="form-row">
          <div class="form-col">
            <div class="form-group">
              <label for="shiftDate">Date</label>
```

```
              <input type="date" id="shiftDate" required>
            </div>
          </div>

          <div class="form-col">
            <div class="form-group">
              <label for="shiftType">Shift</label>
              <select id="shiftType" required>
                <option value="">-- Select Shift --</option>
                <option value="Morning">Morning (8am-4pm)</option>
                <option value="Afternoon">Afternoon (4pm-12am)</option>
                <option value="Night">Night (12am-8am)</option>
              </select>
            </div>
          </div>
        </div>

        <div class="form-group">
          <label for="counselorSelect">Counselor</label>
          <select id="counselorSelect" required>
            <option value="">-- Select Counselor --</option>
            <!-- Counselors will be populated here -->
          </select>
        </div>

        <div class="form-group">
          <label for="shiftNotes">Notes</label>
          <textarea id="shiftNotes" rows="3"></textarea>
        </div>

        <div class="action-buttons">
          <button type="button" class="btn btn-secondary"
onclick="clearShiftForm()">Clear</button>
```

```
          <button type="submit" class="btn btn-primary">Add Shift</button>
        </div>
      </form>
    </div>
  </div>
 </div>
  <!-- Add Shift Modal -->
 <div id="addShiftModal" class="modal">
   <div class="modal-content">
     <button class="modal-close"
onclick="closeAddShiftModal()">&times;</button>
     <h3>Add Counselor to <span id="modalShiftLabel"></span></h3>

     <div class="form-group">
       <label for="modalCounselorSelect">Select Counselor</label>
       <select id="modalCounselorSelect" required>
         <option value="">-- Select Counselor --</option>
         <!-- Counselors will be populated here -->
       </select>
     </div>

     <div class="form-group">
       <label for="modalShiftNotes">Notes</label>
       <textarea id="modalShiftNotes" rows="3"></textarea>
     </div>

     <div class="action-buttons">
       <button type="button" class="btn btn-secondary"
onclick="closeAddShiftModal()">Cancel</button>
       <button type="button" class="btn btn-primary"
onclick="submitModalShift()">Add Shift</button>
     </div>
   </div>
```

```
 </div>
  <!-- Edit Shift Modal -->
 <div id="editShiftModal" class="modal">
   <div class="modal-content">
     <button class="modal-close"
onclick="closeEditShiftModal()">&times;</button>
     <h3>Edit Shift</h3>

     <div class="form-group">
       <label for="editShiftStatus">Status</label>
       <select id="editShiftStatus">
         <option value="Scheduled">Scheduled</option>
         <option value="Completed">Completed</option>
         <option value="Cancelled">Cancelled</option>
       </select>
     </div>

     <div class="form-group">
       <label for="editShiftNotes">Notes</label>
       <textarea id="editShiftNotes" rows="3"></textarea>
     </div>

     <div class="action-buttons">
       <button type="button" class="btn btn-secondary"
onclick="closeEditShiftModal()">Cancel</button>
       <button type="button" class="btn btn-primary"
onclick="updateShift()">Update Shift</button>
       <button type="button" class="btn btn-secondary" onclick="deleteShift()"
style="margin-left: auto;">Delete</button>
     </div>
   </div>
 </div>
  <script>
```

```
  // Global variables
  let currentWeekStart = null;
  let counselors = [];
  let currentShifts = [];
  let modalDate = null;
  let modalShift = null;
  let editingShiftId = null;

  // Initialize the page
  function initialize() {
    // Set up tabs
    document.querySelectorAll('.tab').forEach(function(tab) {
      tab.addEventListener('click', function() {
        // Get tab id
        const tabId = tab.getAttribute('data-tab');

        // Update active tab
        document.querySelectorAll('.tab').forEach(tab =>
tab.classList.remove('active'));
        tab.classList.add('active');

        // Show corresponding content
        document.querySelectorAll('.tab-content').forEach(content =>
content.classList.remove('active'));
        document.getElementById(tabId).classList.add('active');

        // Load specific tab data
        if (tabId === 'list-view') {
          loadListView();
        }
      });
    });
```

```
    // Set current week to this week
    goToToday();

    // Set up form submission
    document.getElementById('addShiftForm').addEventListener('submit',
function(event) {
      event.preventDefault();
      addShift();
    });

    // Set default date to today
    document.getElementById('shiftDate').value = new
Date().toISOString().split('T')[0];

    // Load counselors
    loadCounselors();

    // Set up list view filter change handler
    document.getElementById('listViewFilter').addEventListener('change',
function() {
      const filterValue = this.value;
      document.getElementById('counselorFilterContainer').style.display =
        filterValue === 'counselor' ? 'block' : 'none';
    });
  }

  // Go to today's week
  function goToToday() {
    const today = new Date();
    setCurrentWeek(today);
  }

  // Previous week
```

```
  function previousWeek() {
    const newDate = new Date(currentWeekStart);
    newDate.setDate(newDate.getDate() - 7);
    setCurrentWeek(newDate);
  }

  // Next week
  function nextWeek() {
    const newDate = new Date(currentWeekStart);
    newDate.setDate(newDate.getDate() + 7);
    setCurrentWeek(newDate);
  }

  // Set current week
  function setCurrentWeek(date) {
    // Get the start of the week (Sunday)
    const weekStart = new Date(date);
    weekStart.setDate(date.getDate() - date.getDay());

    // Set the current week start
    currentWeekStart = new Date(weekStart);

    // Calculate end of week
    const weekEnd = new Date(weekStart);
    weekEnd.setDate(weekStart.getDate() + 6);

    // Format dates for display
    const startFormatted = formatDate(weekStart);
    const endFormatted = formatDate(weekEnd);

    // Update the displayed week
    document.getElementById('currentWeekLabel').textContent =
`${startFormatted} - ${endFormatted}`;
```

```
  // Load shifts for this week
  loadWeekShifts();
}

// Format date as Month Day, Year
function formatDate(date) {
  const options = { month: 'long', day: 'numeric', year: 'numeric' };
  return date.toLocaleDateString(undefined, options);
}

// Load counselors
function loadCounselors() {
  google.script.run
    .withSuccessHandler(function(result) {
      counselors = result;

      // Populate counselor selects
      populateCounselorSelects();

      // Populate counselor filter
      populateCounselorFilter();
    })
    .withFailureHandler(function(error) {
      showError('Failed to load counselors: ' + error);
    })
    .getTeamMembers();
}

// Populate counselor selects
function populateCounselorSelects() {
  const selects = [
    document.getElementById('counselorSelect'),
```

```
    document.getElementById('modalCounselorSelect')
  ];

  selects.forEach(function(select) {
    // Clear existing options except the first one
    while (select.options.length > 1) {
      select.remove(1);
    }

    // Add counselors
    counselors.forEach(function(counselor) {
      const option = document.createElement('option');
      option.value = counselor.email;
      option.textContent = counselor.name;
      select.appendChild(option);
    });
  });
}

// Populate counselor filter
function populateCounselorFilter() {
  const select = document.getElementById('counselorFilter');

  // Clear existing options except the first one
  while (select.options.length > 1) {
    select.remove(1);
  }

  // Add counselors
  counselors.forEach(function(counselor) {
    const option = document.createElement('option');
    option.value = counselor.email;
    option.textContent = counselor.name;
```

```
    select.appendChild(option);
  });
}

// Load shifts for the current week
function loadWeekShifts() {
  // Show loading
  document.getElementById('weekViewLoading').style.display = 'block';
  document.getElementById('scheduleGrid').style.display = 'none';

  // Calculate end of week
  const weekEnd = new Date(currentWeekStart);
  weekEnd.setDate(currentWeekStart.getDate() + 6);

  // Format dates for API
  const startStr = currentWeekStart.toISOString().split('T')[0];
  const endStr = weekEnd.toISOString().split('T')[0];

  google.script.run
    .withSuccessHandler(function(shifts) {
      currentShifts = shifts;
      renderWeekView(shifts);

      document.getElementById('weekViewLoading').style.display = 'none';
      document.getElementById('scheduleGrid').style.display = 'grid';
    })
    .withFailureHandler(function(error) {
      showError('Failed to load shifts: ' + error);
      document.getElementById('weekViewLoading').style.display = 'none';
    })
    .getShifts(startStr, endStr);
}
```

```
// Render the week view
function renderWeekView(shifts) {
  // Clear all cells
  const days = ['sun', 'mon', 'tue', 'wed', 'thu', 'fri', 'sat'];
  const shiftTypes = ['morning', 'afternoon', 'night'];

  days.forEach(function(day) {
    shiftTypes.forEach(function(shift) {
      const cell = document.getElementById(`${day}-${shift}`);

      // Clear existing content
      while (cell.firstChild) {
        cell.removeChild(cell.firstChild);
      }

      // Add the add button
      const addButton = document.createElement('div');
      addButton.className = 'add-counselor';
      addButton.textContent = '+';
      addButton.addEventListener('click', function() {
        showAddShiftModal(day, shift);
      });

      cell.appendChild(addButton);

      // Remove today class
      cell.classList.remove('today');
    });
  });

  // Get today's day of week
  const today = new Date();
  const todayDay = today.getDay(); // 0 = Sunday, 6 = Saturday
```

```
// Check if today is in the current week
const weekStart = new Date(currentWeekStart);
const weekEnd = new Date(weekStart);
weekEnd.setDate(weekStart.getDate() + 6);

if (today >= weekStart && today <= weekEnd) {
  // Add today class to today's cells
  const todayKey = days[todayDay];
  shiftTypes.forEach(function(shift) {
    const cell = document.getElementById(`${todayKey}-${shift}`);
    cell.classList.add('today');
  });
}

// Populate shifts
shifts.forEach(function(shift) {
  const shiftDate = new Date(shift.date);
  const dayOfWeek = shiftDate.getDay(); // 0 = Sunday, 6 = Saturday
  const dayKey = days[dayOfWeek];

  // Convert shift type to lowercase
  const shiftType = shift.shiftType.toLowerCase();

  // Check if this is a valid shift type
  if (shiftTypes.includes(shiftType)) {
    const cell = document.getElementById(`${dayKey}-${shiftType}`);

    // Create counselor slot
    const counselorSlot = document.createElement('div');
    counselorSlot.className = 'counselor-slot';
    counselorSlot.textContent = shift.counselorName;
    counselorSlot.setAttribute('data-shift-id', shift.id);
```

```
        // Add click event to edit
        counselorSlot.addEventListener('click', function() {
          showEditShiftModal(shift.id);
        });

        // Insert before the add button
        cell.insertBefore(counselorSlot, cell.lastChild);
      }
    });
  }

  // Show add shift modal
  function showAddShiftModal(day, shift) {
    // Calculate the date for this cell
    const cellDate = new Date(currentWeekStart);
    const dayIndex = ['sun', 'mon', 'tue', 'wed', 'thu', 'fri',
'sat'].indexOf(day);
    cellDate.setDate(cellDate.getDate() + dayIndex);

    // Store the date and shift
    modalDate = cellDate;
    modalShift = shift.charAt(0).toUpperCase() + shift.slice(1); //
Capitalize first letter

    // Update modal label
    document.getElementById('modalShiftLabel').textContent =
      `${formatDate(cellDate)} - ${modalShift} Shift`;

    // Clear form
    document.getElementById('modalCounselorSelect').value = '';
    document.getElementById('modalShiftNotes').value = '';
```

```
    // Show modal
    document.getElementById('addShiftModal').style.display = 'flex';
  }

  // Close add shift modal
  function closeAddShiftModal() {
    document.getElementById('addShiftModal').style.display = 'none';
  }

  // Submit modal shift
  function submitModalShift() {
    const counselorEmail =
document.getElementById('modalCounselorSelect').value;
    const notes = document.getElementById('modalShiftNotes').value;

    if (!counselorEmail) {
      alert('Please select a counselor');
      return;
    }

    // Create shift data
    const shiftData = {
      date: modalDate.toISOString().split('T')[0],
      shiftType: modalShift,
      counselorEmail: counselorEmail,
      notes: notes
    };

    // Submit to server
    google.script.run
      .withSuccessHandler(function(result) {
        if (result.success) {
          showSuccess('Shift added successfully');
```

```
            closeAddShiftModal();
            loadWeekShifts(); // Reload the week view
          } else {
            showError(result.message || 'Failed to add shift');
          }
        })
        .withFailureHandler(function(error) {
          showError('Error adding shift: ' + error);
        })
        .addShift(shiftData);
    }

    // Show edit shift modal
    function showEditShiftModal(shiftId) {
      // Find the shift
      const shift = currentShifts.find(s => s.id === shiftId);

      if (!shift) {
        showError('Shift not found');
        return;
      }

      // Store the shift ID
      editingShiftId = shiftId;

      // Update form
      document.getElementById('editShiftStatus').value = shift.status ||
'Scheduled';
      document.getElementById('editShiftNotes').value = shift.notes || '';

      // Show modal
      document.getElementById('editShiftModal').style.display = 'flex';
    }
```

```
// Close edit shift modal
function closeEditShiftModal() {
  document.getElementById('editShiftModal').style.display = 'none';
  editingShiftId = null;
}

// Update shift
function updateShift() {
  const status = document.getElementById('editShiftStatus').value;
  const notes = document.getElementById('editShiftNotes').value;

  // Create update data
  const updateData = {
    id: editingShiftId,
    status: status,
    notes: notes
  };

  // Submit to server
  google.script.run
    .withSuccessHandler(function(result) {
      if (result.success) {
        showSuccess('Shift updated successfully');
        closeEditShiftModal();
        loadWeekShifts(); // Reload the week view
      } else {
        showError(result.message || 'Failed to update shift');
      }
    })
    .withFailureHandler(function(error) {
      showError('Error updating shift: ' + error);
    })
```

```
    .updateShift(updateData);
}

// Delete shift
function deleteShift() {
  if (confirm('Are you sure you want to delete this shift?')) {
    google.script.run
      .withSuccessHandler(function(result) {
        if (result.success) {
          showSuccess('Shift deleted successfully');
          closeEditShiftModal();
          loadWeekShifts(); // Reload the week view
        } else {
          showError(result.message || 'Failed to delete shift');
        }
      })
      .withFailureHandler(function(error) {
        showError('Error deleting shift: ' + error);
      })
      .deleteShift(editingShiftId);
  }
}

// Add shift from the form
function addShift() {
  const date = document.getElementById('shiftDate').value;
  const shiftType = document.getElementById('shiftType').value;
  const counselorEmail = document.getElementById('counselorSelect').value;
  const notes = document.getElementById('shiftNotes').value;

  // Validate required fields
  if (!date || !shiftType || !counselorEmail) {
    showError('Please fill out all required fields');
```

```
      return;
    }

    // Create shift data
    const shiftData = {
      date: date,
      shiftType: shiftType,
      counselorEmail: counselorEmail,
      notes: notes
    };

    // Submit to server
    google.script.run
      .withSuccessHandler(function(result) {
        if (result.success) {
          showSuccess('Shift added successfully');
          clearShiftForm();

          // Reload the current view
          const activeTab =
document.querySelector('.tab.active').getAttribute('data-tab');
          if (activeTab === 'week-view') {
            loadWeekShifts();
          } else if (activeTab === 'list-view') {
            loadListView();
          }
        } else {
          showError(result.message || 'Failed to add shift');
        }
      })
      .withFailureHandler(function(error) {
        showError('Error adding shift: ' + error);
      })
```

```
      .addShift(shiftData);
  }

  // Clear shift form
  function clearShiftForm() {
    document.getElementById('shiftDate').value = new
Date().toISOString().split('T')[0];
    document.getElementById('shiftType').value = '';
    document.getElementById('counselorSelect').value = '';
    document.getElementById('shiftNotes').value = '';
  }

  // Load list view
  function loadListView() {
    // Show loading
    document.getElementById('listViewLoading').style.display = 'block';
    document.getElementById('shiftsList').innerHTML = '';

    google.script.run
      .withSuccessHandler(function(shifts) {
        currentShifts = shifts;
        filterListView();

        document.getElementById('listViewLoading').style.display = 'none';
      })
      .withFailureHandler(function(error) {
        showError('Failed to load shifts: ' + error);
        document.getElementById('listViewLoading').style.display = 'none';
      })
      .getAllShifts();
  }

  // Filter list view
```

```
function filterListView() {
  const filterType = document.getElementById('listViewFilter').value;
  const counselorEmail = document.getElementById('counselorFilter').value;
  const today = new Date();
  today.setHours(0, 0, 0, 0); // Set to start of day

  // Apply filters
  let filteredShifts = [...currentShifts];

  if (filterType === 'future') {
    filteredShifts = filteredShifts.filter(shift => {
      const shiftDate = new Date(shift.date);
      return shiftDate >= today;
    });
  } else if (filterType === 'past') {
    filteredShifts = filteredShifts.filter(shift => {
      const shiftDate = new Date(shift.date);
      return shiftDate < today;
    });
  } else if (filterType === 'counselor' && counselorEmail) {
    filteredShifts = filteredShifts.filter(shift =>
      shift.counselorEmail === counselorEmail
    );
  }

  // Sort by date (newest first)
  filteredShifts.sort((a, b) => new Date(b.date) - new Date(a.date));

  // Render the filtered shifts
  renderListView(filteredShifts);
}

// Render list view
```

```
  function renderListView(shifts) {
    const listElement = document.getElementById('shiftsList');
    listElement.innerHTML = '';

    if (shifts.length === 0) {
      listElement.innerHTML = '<div style="text-align: center; padding:
20px;">No shifts found.</div>';
      return;
    }

    shifts.forEach(function(shift) {
      const shiftDate = new Date(shift.date);
      const dateStr = formatDate(shiftDate);

      const listItem = document.createElement('div');
      listItem.className = 'list-view-item';

      // Determine status badge background
      let statusBg = 'var(--soft-lavender)';
      if (shift.status === 'Completed') {
        statusBg = 'var(--soft-green)';
      } else if (shift.status === 'Cancelled') {
        statusBg = 'var(--light-pink)';
      }

      listItem.innerHTML = `
        <div class="counselor-info">
          <div class="counselor-name">${shift.counselorName}</div>
          <div class="counselor-date">${dateStr} - ${shift.shiftType}
Shift</div>
        </div>
        <div class="status-badge" style="background-color:
${statusBg};">${shift.status || 'Scheduled'}</div>
```

```
      `;

      // Add click event to edit
      listItem.addEventListener('click', function() {
        showEditShiftModal(shift.id);
      });

      listElement.appendChild(listItem);
    });
}

// Show success message
function showSuccess(message) {
  const successElement = document.getElementById('successMessage');
  successElement.textContent = message;
  successElement.style.display = 'block';

  // Hide after 3 seconds
  setTimeout(function() {
    successElement.style.display = 'none';
  }, 3000);
}

// Show error message
function showError(message) {
  const errorElement = document.getElementById('errorMessage');
  errorElement.textContent = message;
  errorElement.style.display = 'block';

  // Hide after 5 seconds
  setTimeout(function() {
    errorElement.style.display = 'none';
  }, 5000);
```

```
    }

    // Initialize when the page loads
    google.script.onload = function() {
      initialize();
    };
  </script>
</body>
</html>
<!DOCTYPE html>
<html>
<head>
  <base target="_top">
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link
href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500;700&fam
ily=Poppins:wght@500;600;700&display=swap" rel="stylesheet">
  <style>
    /* Variables for 988 Lifeline LGBTQIA+ Services color scheme */
    :root {
      /* Primary Colors */
      --trevor-orange: #FF786E;
      --deep-blue: #001A4E;
      --purple: #9A3499;
      --teal: #137F6A;

      /* Secondary Colors */
      --light-blue: #4F52DE;
      --soft-yellow: #FFAD8D;
      --lavender: #B3AE4A;
      --light-purple: #F54AC;
```

```
    /* Tertiary Colors */
    --soft-green: #BAE2CE;
    --pale-yellow: #FFF2DF;
    --light-pink: #FBCBBE;
    --soft-lavender: #D1CFCC;

    /* Gradients */
    --primary-gradient: linear-gradient(135deg, var(--trevor-orange),
var(--light-pink));
    --calm-gradient: linear-gradient(135deg, var(--light-blue),
var(--soft-green));
    --support-gradient: linear-gradient(45deg, var(--purple),
var(--light-purple));
    --background-gradient: linear-gradient(180deg, var(--pale-yellow),
#FFFFFF);

    /* Spacing */
    --spacing-xs: 4px;
    --spacing-sm: 8px;
    --spacing-md: 16px;
    --spacing-lg: 24px;
    --spacing-xl: 32px;

    /* Borders */
    --border-radius-sm: 8px;
    --border-radius-md: 12px;
    --border-radius-lg: 16px;
  }

  body {
    font-family: 'Roboto', sans-serif;
    margin: 0;
    padding: 0;
```

```
  background-color: var(--pale-yellow);
  color: var(--deep-blue);
}

.container {
  max-width: 700px;
  margin: 0 auto;
  padding: var(--spacing-md);
}

.header {
  margin-bottom: var(--spacing-lg);
  text-align: center;
}

h1, h2, h3, h4 {
  font-family: 'Poppins', sans-serif;
  color: var(--deep-blue);
}

.tabs {
  display: flex;
  border-bottom: 1px solid #ddd;
  margin-bottom: var(--spacing-lg);
}

.tab {
  padding: var(--spacing-sm) var(--spacing-lg);
  cursor: pointer;
  border-bottom: 2px solid transparent;
  transition: all 0.2s ease;
}
```

```
.tab.active {
  border-bottom-color: var(--trevor-orange);
  font-weight: 500;
}

.tab-content {
  display: none;
}

.tab-content.active {
  display: block;
}

.card {
  background: white;
  border-radius: var(--border-radius-md);
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.05);
  padding: var(--spacing-lg);
  margin-bottom: var(--spacing-lg);
  position: relative;
}

.form-group {
  margin-bottom: var(--spacing-md);
}

label {
  display: block;
  font-weight: 500;
  margin-bottom: var(--spacing-xs);
}

input, select, textarea {
```

```
  width: 100%;
  padding: var(--spacing-sm);
  border: 1px solid #ddd;
  border-radius: var(--border-radius-sm);
  font-family: 'Roboto', sans-serif;
  font-size: 16px;
}

textarea {
  resize: vertical;
  min-height: 100px;
}

.btn {
  border: none;
  padding: var(--spacing-md) var(--spacing-lg);
  border-radius: var(--border-radius-sm);
  font-weight: 500;
  cursor: pointer;
  transition: all 0.2s ease;
}

.btn-primary {
  background: var(--primary-gradient);
  color: white;
}

.btn-primary:hover {
  box-shadow: 0 4px 8px rgba(255, 120, 110, 0.3);
  transform: translateY(-2px);
}

.btn-secondary {
```

```
  background: white;
  color: var(--deep-blue);
  border: 1px solid #ddd;
}

.btn-secondary:hover {
  background: #f5f5f5;
}

.form-actions {
  display: flex;
  justify-content: flex-end;
  gap: var(--spacing-sm);
  margin-top: var(--spacing-lg);
}

.success-message {
  background-color: var(--soft-green);
  color: var(--deep-blue);
  padding: var(--spacing-md);
  border-radius: var(--border-radius-sm);
  margin-bottom: var(--spacing-lg);
  text-align: center;
  display: none;
}

.error-message {
  background-color: var(--light-pink);
  color: var(--deep-blue);
  padding: var(--spacing-md);
  border-radius: var(--border-radius-sm);
  margin-bottom: var(--spacing-lg);
  text-align: center;
```

```
  display: none;
}

.section-title {
  font-family: 'Poppins', sans-serif;
  font-size: 1.1rem;
  margin-bottom: var(--spacing-md);
  padding-bottom: var(--spacing-xs);
  border-bottom: 1px solid #eee;
}

.status-item {
  display: flex;
  justify-content: space-between;
  padding: var(--spacing-sm) 0;
  border-bottom: 1px solid #f0f0f0;
}

.status-item:last-child {
  border-bottom: none;
}

.status-label {
  font-weight: 500;
}

.status-value {
  color: #666;
}

.status-value.ok {
  color: var(--teal);
}
```

```
  .status-value.error {
    color: var(--trevor-orange);
  }

  .help-text {
    font-size: 14px;
    color: #666;
    margin-top: var(--spacing-xs);
  }

  .card::before {
    content: '';
    position: absolute;
    left: 0;
    top: 0;
    height: 100%;
    width: 3px;
  }

  .card.general::before {
    background: var(--trevor-orange);
  }

  .card.integrations::before {
    background: var(--light-blue);
  }

  .card.system::before {
    background: var(--purple);
  }
 </style>
</head>
```

```
<body>
 <div class="container">
   <div class="header">
     <h1>Settings</h1>
     <p>Configure the 988 Lifeline LGBTQIA+ Crisis Support System</p>
   </div>

   <div id="successMessage" class="success-message">
     Settings saved successfully!
   </div>

   <div id="errorMessage" class="error-message">
     An error occurred. Please try again.
   </div>

   <div class="tabs">
     <div class="tab active" data-tab="general">General</div>
     <div class="tab" data-tab="integrations">Integrations</div>
     <div class="tab" data-tab="system">System Status</div>
   </div>

   <!-- General Settings Tab -->
   <div id="general" class="tab-content active">
     <div class="card general">
       <h3>General Settings</h3>

       <div class="form-group">
         <label for="organizationName">Organization Name</label>
         <input type="text" id="organizationName" placeholder="988 Lifeline
LGBTQIA+ Crisis Support">
       </div>

       <div class="form-group">
```

```
        <label for="primaryContact">Primary Contact Email</label>
        <input type="email" id="primaryContact"
placeholder="contact@example.com">
      </div>

      <div class="form-group">
        <label for="timeZone">Time Zone</label>
        <select id="timeZone">
          <option value="America/New_York">Eastern Time (ET)</option>
          <option value="America/Chicago">Central Time (CT)</option>
          <option value="America/Denver">Mountain Time (MT)</option>
          <option value="America/Los_Angeles">Pacific Time (PT)</option>
          <option value="America/Anchorage">Alaska Time (AKT)</option>
          <option value="Pacific/Honolulu">Hawaii Time (HT)</option>
        </select>
      </div>

      <div class="form-actions">
        <button type="button" class="btn btn-primary"
onclick="saveGeneralSettings()">Save Settings</button>
      </div>
    </div>
  </div>

  <!-- Integrations Tab -->
  <div id="integrations" class="tab-content">
    <div class="card integrations">
      <h3>Asana Integration</h3>

      <div class="form-group">
        <label for="asanaApiKey">Asana API Key</label>
        <input type="password" id="asanaApiKey" placeholder="Enter Asana API
key">
```

```
        <div class="help-text">You can find your Asana API key in your Asana account settings</div>
      </div>

      <div class="form-group">
        <label for="asanaWorkspace">Asana Workspace GID</label>
        <input type="text" id="asanaWorkspace" placeholder="Enter workspace GID">
      </div>

      <div class="form-group">
        <label for="asanaProject">Default Asana Project GID</label>
        <input type="text" id="asanaProject" placeholder="Enter project GID">
      </div>

      <div class="form-actions">
        <button type="button" class="btn btn-secondary" onclick="testAsanaConnection()">Test Connection</button>
        <button type="button" class="btn btn-primary" onclick="saveAsanaSettings()">Save Settings</button>
      </div>
    </div>
  </div>

  <!-- System Status Tab -->
  <div id="system" class="tab-content">
    <div class="card system">
      <h3>System Status</h3>

      <div class="section-title">System Information</div>

      <div id="systemInfo">
        <div class="status-item">
```

```
          <div class="status-label">Version</div>
          <div class="status-value" id="systemVersion">Loading...</div>
        </div>
        <div class="status-item">
          <div class="status-label">Initialization Date</div>
          <div class="status-value" id="initDate">Loading...</div>
        </div>
      </div>

      <div class="section-title">Required Sheets</div>

      <div id="sheetStatus">
        <div class="status-item">
          <div class="status-label">Counselor Tracking</div>
          <div class="status-value"
id="counselorSheetStatus">Loading...</div>
        </div>
        <div class="status-item">
          <div class="status-label">Call Metrics</div>
          <div class="status-value" id="metricsSheetStatus">Loading...</div>
        </div>
        <div class="status-item">
          <div class="status-label">Alerts</div>
          <div class="status-value" id="alertsSheetStatus">Loading...</div>
        </div>
        <div class="status-item">
          <div class="status-label">Tasks</div>
          <div class="status-value" id="tasksSheetStatus">Loading...</div>
        </div>
        <div class="status-item">
          <div class="status-label">Schedule</div>
          <div class="status-value" id="scheduleSheetStatus">Loading...</div>
        </div>
```

```
      </div>

      <div class="form-actions">
        <button type="button" class="btn btn-secondary"
onclick="runSystemValidation()">Run Validation</button>
        <button type="button" class="btn btn-primary"
onclick="repairSystem()">Repair System</button>
      </div>
    </div>
  </div>
 </div>
  <script>
   // Initialize the settings page
   function initialize() {
     // Set up tabs
     document.querySelectorAll('.tab').forEach(function(tab) {
       tab.addEventListener('click', function() {
         // Get tab id
         const tabId = tab.getAttribute('data-tab');

         // Update active tab
         document.querySelectorAll('.tab').forEach(tab =>
tab.classList.remove('active'));
         tab.classList.add('active');

         // Show corresponding content
         document.querySelectorAll('.tab-content').forEach(content =>
content.classList.remove('active'));
         document.getElementById(tabId).classList.add('active');

         // Load tab-specific data
         if (tabId === 'general') {
           loadGeneralSettings();
```

```
        } else if (tabId === 'integrations') {
          loadAsanaSettings();
        } else if (tabId === 'system') {
          loadSystemStatus();
        }
      });
    });

    // Load initial tab data
    loadGeneralSettings();
  }

  // Load general settings
  function loadGeneralSettings() {
    google.script.run
      .withSuccessHandler(function(settings) {
        document.getElementById('organizationName').value =
settings.organizationName || '';
        document.getElementById('primaryContact').value =
settings.primaryContact || '';
        document.getElementById('timeZone').value = settings.timeZone ||
'America/New_York';
      })
      .withFailureHandler(function(error) {
        showError('Failed to load general settings: ' + error);
      })
      .getGeneralSettings();
  }

  // Save general settings
  function saveGeneralSettings() {
    const settings = {
      organizationName: document.getElementById('organizationName').value,
```

```
      primaryContact: document.getElementById('primaryContact').value,
      timeZone: document.getElementById('timeZone').value
    };

    google.script.run
      .withSuccessHandler(function(result) {
        if (result.success) {
          showSuccess('General settings saved successfully');
        } else {
          showError(result.message || 'Failed to save general settings');
        }
      })
      .withFailureHandler(function(error) {
        showError('Error saving general settings: ' + error);
      })
      .saveGeneralSettings(settings);
  }

  // Load Asana settings
  function loadAsanaSettings() {
    google.script.run
      .withSuccessHandler(function(settings) {
        document.getElementById('asanaApiKey').value = settings.apiKey || '';
        document.getElementById('asanaWorkspace').value =
settings.workspaceGid || '';
        document.getElementById('asanaProject').value = settings.projectGid
|| '';
      })
      .withFailureHandler(function(error) {
        showError('Failed to load Asana settings: ' + error);
      })
      .getAsanaSettings();
  }
```

```
// Save Asana settings
function saveAsanaSettings() {
  const settings = {
    apiKey: document.getElementById('asanaApiKey').value,
    workspaceGid: document.getElementById('asanaWorkspace').value,
    projectGid: document.getElementById('asanaProject').value
  };

  google.script.run
    .withSuccessHandler(function(result) {
      if (result.success) {
        showSuccess('Asana settings saved successfully');
      } else {
        showError(result.message || 'Failed to save Asana settings');
      }
    })
    .withFailureHandler(function(error) {
      showError('Error saving Asana settings: ' + error);
    })
    .saveAsanaSettings(settings);
}

// Test Asana connection
function testAsanaConnection() {
  const apiKey = document.getElementById('asanaApiKey').value;
  const workspaceGid = document.getElementById('asanaWorkspace').value;

  if (!apiKey || !workspaceGid) {
    showError('Please enter API key and workspace GID');
    return;
  }
```

```
    google.script.run
      .withSuccessHandler(function(result) {
        if (result.success) {
          showSuccess('Asana connection successful');
        } else {
          showError(result.message || 'Asana connection failed');
        }
      })
      .withFailureHandler(function(error) {
        showError('Error testing Asana connection: ' + error);
      })
      .testAsanaConnection(apiKey, workspaceGid);
  }

  // Load system status
  function loadSystemStatus() {
    // Load system info
    google.script.run
      .withSuccessHandler(function(info) {
        document.getElementById('systemVersion').textContent = info.version
|| 'Unknown';
        document.getElementById('initDate').textContent = info.initDate ||
'Not initialized';
      })
      .withFailureHandler(function(error) {
        console.error('Failed to load system info:', error);
        document.getElementById('systemVersion').textContent = 'Error';
        document.getElementById('initDate').textContent = 'Error';
      })
      .getSystemInfo();

    // Load sheet status
    google.script.run
```

```
      .withSuccessHandler(function(status) {
        // Update sheet status
        updateSheetStatus('counselorSheetStatus', status.counselorTracking);
        updateSheetStatus('metricsSheetStatus', status.callMetrics);
        updateSheetStatus('alertsSheetStatus', status.alerts);
        updateSheetStatus('tasksSheetStatus', status.tasks);
        updateSheetStatus('scheduleSheetStatus', status.schedule);
      })
      .withFailureHandler(function(error) {
        console.error('Failed to load sheet status:', error);
        document.querySelectorAll('#sheetStatus
.status-value').forEach(element => {
          element.textContent = 'Error';
          element.className = 'status-value error';
        });
      })
      .getSheetStatus();
  }

  // Update sheet status display
  function updateSheetStatus(elementId, status) {
    const element = document.getElementById(elementId);

    if (status) {
      element.textContent = 'OK';
      element.className = 'status-value ok';
    } else {
      element.textContent = 'Missing';
      element.className = 'status-value error';
    }
  }

  // Run system validation
```

```
   function runSystemValidation() {
     google.script.run
       .withSuccessHandler(function(result) {
         if (result.valid) {
           showSuccess('System validation completed successfully. All
components are valid.');
         } else {
           let message = 'System validation failed. The following issues were
found:';

           if (result.missingSheets && result.missingSheets.length > 0) {
             message += '\n\nMissing sheets: ' + result.missingSheets.join(',
');
           }

           if (result.headerIssues && result.headerIssues.length > 0) {
             message += '\n\nHeader issues:\n' +
result.headerIssues.join('\n');
           }

           if (result.missingProperties && result.missingProperties.length >
0) {
             message += '\n\nMissing properties: ' +
result.missingProperties.join(', ');
           }

           // Use confirm to show multiline message with repair option
           if (confirm(message + '\n\nWould you like to repair these
issues?')) {
             repairSystem();
           }
         }
```

```
        // Reload status after validation
        loadSystemStatus();
      })
      .withFailureHandler(function(error) {
        showError('Error running system validation: ' + error);
      })
      .validateSystem();
  }

  // Repair system
  function repairSystem() {
    google.script.run
      .withSuccessHandler(function(result) {
        if (result.success) {
          showSuccess('System repair completed successfully.');
        } else {
          showError('Some system repair operations failed: ' + (result.error
|| 'Unknown error'));
        }

        // Reload status after repair
        loadSystemStatus();
      })
      .withFailureHandler(function(error) {
        showError('Error repairing system: ' + error);
      })
      .repairSystem();
  }

  // Show success message
  function showSuccess(message) {
    const successElement = document.getElementById('successMessage');
    successElement.textContent = message;
```

```
    successElement.style.display = 'block';

    // Hide after 3 seconds
    setTimeout(function() {
      successElement.style.display = 'none';
    }, 3000);
  }

  // Show error message
  function showError(message) {
    const errorElement = document.getElementById('errorMessage');
    errorElement.textContent = message;
    errorElement.style.display = 'block';

    // Hide after 5 seconds
    setTimeout(function() {
      errorElement.style.display = 'none';
    }, 5000);
  }

  // Initialize when the page loads
  google.script.onload = function() {
    initialize();
  };
 </script>
</body>
</html>
<!DOCTYPE html>
<html>
<head>
 <base target="_top">
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
 <link
href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500;700&fam
ily=Poppins:wght@500;600;700&display=swap" rel="stylesheet">
 <style>
   /* Variables for Trevor Project color scheme */
   :root {
     /* Primary Colors */
     --trevor-orange: #FF786E;
     --deep-blue: #001A4E;
     --purple: #9A3499;
     --teal: #137F6A;

     /* Secondary Colors */
     --light-blue: #4F52DE;
     --soft-yellow: #FFAD8D;
     --lavender: #B3AE4A;
     --light-purple: #F54AC;

     /* Tertiary Colors */
     --soft-green: #BAE2CE;
     --pale-yellow: #FFF2DF;
     --light-pink: #FBCBBE;
     --soft-lavender: #D1CFCC;

     /* Gradients */
     --primary-gradient: linear-gradient(135deg, var(--trevor-orange),
var(--light-pink));
     --calm-gradient: linear-gradient(135deg, var(--light-blue),
var(--soft-green));
     --support-gradient: linear-gradient(45deg, var(--purple),
var(--light-purple));
     --background-gradient: linear-gradient(180deg, var(--pale-yellow),
#FFFFFF);
```

```
  /* Spacing */
  --spacing-xs: 4px;
  --spacing-sm: 8px;
  --spacing-md: 16px;
  --spacing-lg: 24px;
  --spacing-xl: 32px;

  /* Borders */
  --border-radius-sm: 8px;
  --border-radius-md: 12px;
  --border-radius-lg: 16px;
}

body {
  font-family: 'Roboto', sans-serif;
  margin: 0;
  padding: 0;
  background-color: var(--pale-yellow);
  color: var(--deep-blue);
}

.container {
  max-width: 800px;
  margin: 0 auto;
  padding: var(--spacing-md);
}

h1, h2, h3 {
  font-family: 'Poppins', sans-serif;
  color: var(--deep-blue);
}
```

```css
.header {
  margin-bottom: var(--spacing-lg);
  text-align: center;
}

.form-card {
  background: white;
  border-radius: var(--border-radius-md);
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.05);
  padding: var(--spacing-lg);
  margin-bottom: var(--spacing-lg);
  position: relative;
}

.form-card::before {
  content: '';
  position: absolute;
  left: 0;
  top: 0;
  height: 100%;
  width: 3px;
  background: var(--trevor-orange);
}

.form-group {
  margin-bottom: var(--spacing-md);
}

label {
  display: block;
  font-weight: 500;
  margin-bottom: var(--spacing-xs);
}
```

```css
input, select, textarea {
  width: 100%;
  padding: var(--spacing-sm);
  border: 1px solid #ddd;
  border-radius: var(--border-radius-sm);
  font-family: 'Roboto', sans-serif;
  font-size: 16px;
}

textarea {
  resize: vertical;
  min-height: 100px;
}

.goals-container {
  margin-bottom: var(--spacing-md);
}

.goal-input {
  display: flex;
  margin-bottom: var(--spacing-sm);
}

.goal-input input {
  flex-grow: 1;
}

.goal-input button {
  margin-left: var(--spacing-sm);
}

.btn {
```

```
  border: none;
  padding: var(--spacing-md) var(--spacing-lg);
  border-radius: var(--border-radius-sm);
  font-weight: 500;
  cursor: pointer;
  transition: all 0.2s ease;
}

.btn-primary {
  background: var(--primary-gradient);
  color: white;
}

.btn-primary:hover {
  box-shadow: 0 4px 8px rgba(255, 120, 110, 0.3);
  transform: translateY(-2px);
}

.btn-secondary {
  background: white;
  color: var(--deep-blue);
  border: 1px solid #ddd;
}

.btn-secondary:hover {
  background: #f5f5f5;
}

.btn-sm {
  font-size: 14px;
  padding: var(--spacing-xs) var(--spacing-sm);
}
```

```
    .btn-icon {
      display: flex;
      align-items: center;
      justify-content: center;
      width: 32px;
      height: 32px;
      border-radius: 50%;
      padding: 0;
    }

    .form-actions {
      display: flex;
      justify-content: space-between;
      margin-top: var(--spacing-lg);
    }

    .success-message {
      background-color: var(--soft-green);
      border-radius: var(--border-radius-sm);
      padding: var(--spacing-md);
      margin-bottom: var(--spacing-md);
      display: none;
    }

    .error-message {
      background-color: var(--light-pink);
      border-radius: var(--border-radius-sm);
      padding: var(--spacing-md);
      margin-bottom: var(--spacing-md);
      display: none;
    }
  </style>
</head>
```

```
<body>
 <div class="container">
   <div class="header">
     <h1>Initialize Team Lead Shift</h1>
     <p>Set up your shift goals and priorities to stay focused and track your
progress.</p>
   </div>

   <div id="successMessage" class="success-message">
     Shift initialized successfully!
   </div>

   <div id="errorMessage" class="error-message">
     An error occurred while initializing the shift. Please try again.
   </div>

   <div class="form-card">
     <form id="shiftInitForm">
       <div class="form-group">
         <label for="shiftDate">Shift Date</label>
         <input type="date" id="shiftDate" name="shiftDate" required>
       </div>

       <div class="form-group">
         <label for="shiftType">Shift Type</label>
         <select id="shiftType" name="shiftType" required>
           <option value="">Select shift type</option>
           <option value="Morning">Morning (8am-4pm)</option>
           <option value="Afternoon">Afternoon (4pm-12am)</option>
           <option value="Night">Night (12am-8am)</option>
           <option value="Split">Split Shift</option>
           <option value="Weekend">Weekend</option>
         </select>
```

```
      </div>

      <div class="form-group">
        <label for="startTime">Start Time</label>
        <input type="time" id="startTime" name="startTime" required>
      </div>

      <div class="form-group">
        <label for="endTime">End Time</label>
        <input type="time" id="endTime" name="endTime" required>
      </div>

      <div class="form-group">
        <label>Shift Goals (3-5 recommended)</label>
        <div id="goalsContainer" class="goals-container">
          <div class="goal-input">
            <input type="text" name="goals[]" placeholder="Enter a goal for
this shift" required>
            <button type="button" class="btn btn-secondary btn-icon"
onclick="addGoalInput()">+</button>
          </div>
        </div>
      </div>

      <div class="form-group">
        <label for="quickNotes">Quick Notes & Priorities</label>
        <textarea id="quickNotes" name="quickNotes" placeholder="Enter any
notes or priority items for this shift"></textarea>
      </div>

      <div class="form-actions">
        <button type="button" class="btn btn-secondary"
onclick="google.script.host.close()">Cancel</button>
```

```
        <div>
          <button type="submit" class="btn btn-primary">Initialize
Shift</button>
          <button type="button" class="btn btn-secondary"
id="startTrackerBtn" style="display: none;" onclick="startTimeTracker()">Start
Time Tracker</button>
        </div>
      </div>
    </form>
  </div>
 </div>
  <script>
   // Initialize date field with today's date
   document.addEventListener('DOMContentLoaded', function() {
     const today = new Date().toISOString().split('T')[0];
     document.getElementById('shiftDate').value = today;

     // Set default times based on current time
     const now = new Date();
     const hours = now.getHours();
     let shiftType = '';
     let startTime = '';
     let endTime = '';

     if (hours >= 6 && hours < 14) {
       shiftType = 'Morning';
       startTime = '08:00';
       endTime = '16:00';
     } else if (hours >= 14 && hours < 22) {
       shiftType = 'Afternoon';
       startTime = '16:00';
       endTime = '00:00';
     } else {
```

```
      shiftType = 'Night';
      startTime = '00:00';
      endTime = '08:00';
    }

    document.getElementById('shiftType').value = shiftType;
    document.getElementById('startTime').value = startTime;
    document.getElementById('endTime').value = endTime;

    // Add form submit handler
    document.getElementById('shiftInitForm').addEventListener('submit',
function(e) {
      e.preventDefault();
      initializeShift();
    });
  });

  // Add a new goal input field
  function addGoalInput() {
    const container = document.getElementById('goalsContainer');
    const inputs = container.querySelectorAll('input[name="goals[]"]');

    // Limit to 5 goals
    if (inputs.length >= 5) {
      alert('Maximum of 5 goals allowed');
      return;
    }

    const div = document.createElement('div');
    div.className = 'goal-input';

    const input = document.createElement('input');
    input.type = 'text';
```

```
    input.name = 'goals[]';
    input.placeholder = 'Enter a goal for this shift';

    const button = document.createElement('button');
    button.type = 'button';
    button.className = 'btn btn-secondary btn-icon';
    button.textContent = '-';
    button.onclick = function() {
      container.removeChild(div);
    };

    div.appendChild(input);
    div.appendChild(button);
    container.appendChild(div);
  }

  // Initialize the shift
  function initializeShift() {
    // Get form data
    const form = document.getElementById('shiftInitForm');
    const formData = new FormData(form);

    // Convert to object
    const shiftData = {
      date: formData.get('shiftDate'),
      shiftType: formData.get('shiftType'),
      startTime: formData.get('startTime'),
      endTime: formData.get('endTime'),
      goals: Array.from(formData.getAll('goals[]')).filter(goal =>
goal.trim() !== ''),
      quickNotes: formData.get('quickNotes')
    };
```

```
    // Validate data
    if (shiftData.goals.length === 0) {
      alert('Please enter at least one goal for the shift');
      return;
    }

    // Show loading state
    const submitButton = form.querySelector('button[type="submit"]');
    const originalText = submitButton.textContent;
    submitButton.textContent = 'Initializing...';
    submitButton.disabled = true;

    // Submit to the server
    google.script.run
      .withSuccessHandler(function(result) {
        // Reset button state
        submitButton.textContent = originalText;
        submitButton.disabled = false;

        if (result.success) {
          // Show success message
          document.getElementById('successMessage').style.display = 'block';
          document.getElementById('errorMessage').style.display = 'none';

          // Disable form
          const inputs = form.querySelectorAll('input, select, textarea,
button[type="button"]');
          inputs.forEach(input => input.disabled = true);

          // Show time tracker button
          document.getElementById('startTrackerBtn').style.display =
'inline-block';
        } else {
```

```
            // Show error message
            document.getElementById('errorMessage').style.display = 'block';
            document.getElementById('successMessage').style.display = 'none';
            document.getElementById('errorMessage').textContent =
result.message || 'An error occurred while initializing the shift. Please try
again.';
          }
        })
        .withFailureHandler(function(error) {
          // Reset button state
          submitButton.textContent = originalText;
          submitButton.disabled = false;

          // Show error message
          document.getElementById('errorMessage').style.display = 'block';
          document.getElementById('successMessage').style.display = 'none';
          document.getElementById('errorMessage').textContent = error.message
|| 'An error occurred while initializing the shift. Please try again.';
        })
        .initializeShift(shiftData);
    }

    // Start the time tracker
    function startTimeTracker() {
      google.script.host.close();
      google.script.run.showTimeTracker();
    }
  </script>
</body>
</html>
<!DOCTYPE html>
<html>
<head>
```

```
 <base target="_top">
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <link
href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500;700&fam
ily=Poppins:wght@500;600;700&display=swap" rel="stylesheet">
 <style>
   /* Variables for Trevor Project color scheme */
   :root {
     /* Primary Colors */
     --trevor-orange: #FF786E;
     --deep-blue: #001A4E;
     --purple: #9A3499;
     --teal: #137F6A;

     /* Secondary Colors */
     --light-blue: #4F52DE;
     --soft-yellow: #FFAD8D;
     --lavender: #B3AE4A;
     --light-purple: #F54AC;

     /* Tertiary Colors */
     --soft-green: #BAE2CE;
     --pale-yellow: #FFF2DF;
     --light-pink: #FBCBBE;
     --soft-lavender: #D1CFCC;

     /* Gradients */
     --primary-gradient: linear-gradient(135deg, var(--trevor-orange),
var(--light-pink));
     --calm-gradient: linear-gradient(135deg, var(--light-blue),
var(--soft-green));
```

```
    --support-gradient: linear-gradient(45deg, var(--purple),
var(--light-purple));
    --background-gradient: linear-gradient(180deg, var(--pale-yellow),
#FFFFFF);

    /* Spacing */
    --spacing-xs: 4px;
    --spacing-sm: 8px;
    --spacing-md: 16px;
    --spacing-lg: 24px;
    --spacing-xl: 32px;

    /* Border Radius */
    --border-radius-sm: 8px;
    --border-radius-md: 12px;
    --border-radius-lg: 16px;
  }

  body {
    font-family: 'Roboto', sans-serif;
    margin: 0;
    padding: 0;
    background: var(--pale-yellow);
    color: var(--deep-blue);
    min-height: 100vh;
    font-size: 14px;
  }

  h1, h2, h3, h4, h5, h6 {
    font-family: 'Poppins', sans-serif;
    color: var(--deep-blue);
    margin-top: 0;
  }
```

```css
.sidebar-container {
  display: flex;
  flex-direction: column;
  height: 100vh;
}

.sidebar-header {
  background: var(--deep-blue);
  color: white;
  padding: var(--spacing-md);
  text-align: center;
  position: sticky;
  top: 0;
}

.sidebar-header h1 {
  color: white;
  font-size: 18px;
  margin: 0;
}

.sidebar-content {
  flex: 1;
  overflow-y: auto;
  padding: var(--spacing-md);
}

.sidebar-footer {
  padding: var(--spacing-md);
  background: rgba(0, 0, 0, 0.05);
  border-top: 1px solid rgba(0, 0, 0, 0.1);
  text-align: center;
```

```css
}

.card {
  background: white;
  border-radius: var(--border-radius-md);
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.05);
  margin-bottom: var(--spacing-md);
  overflow: hidden;
  position: relative;
}

.card::before {
  content: '';
  position: absolute;
  left: 0;
  top: 0;
  height: 100%;
  width: 3px;
  background: var(--trevor-orange);
}

.card-header {
  padding: var(--spacing-sm);
  background: rgba(255, 120, 110, 0.05);
  display: flex;
  justify-content: space-between;
  align-items: center;
  font-weight: 500;
}

.card-content {
  padding: var(--spacing-sm);
}
```

```css
.metrics-grid {
  display: grid;
  grid-template-columns: 1fr 1fr;
  gap: var(--spacing-xs);
  margin-bottom: var(--spacing-xs);
}

.metric {
  text-align: center;
  padding: var(--spacing-xs);
}

.metric-value {
  font-size: 18px;
  font-weight: 700;
  margin-bottom: 2px;
}

.metric-label {
  font-size: 12px;
  color: #666;
}

.quick-links {
  list-style: none;
  padding: 0;
  margin: 0;
}

.quick-link {
  padding: var(--spacing-sm);
  border-bottom: 1px solid #eee;
```

```
  display: flex;
  align-items: center;
  cursor: pointer;
  transition: background 0.2s;
}

.quick-link:last-child {
  border-bottom: none;
}

.quick-link:hover {
  background: rgba(255, 120, 110, 0.05);
}

.quick-link-icon {
  margin-right: var(--spacing-sm);
  color: var(--trevor-orange);
  display: inline-flex;
  align-items: center;
  justify-content: center;
  width: 24px;
  height: 24px;
  font-size: 16px;
}

.alert-count {
  background: var(--light-purple);
  color: white;
  border-radius: 50%;
  display: inline-flex;
  align-items: center;
  justify-content: center;
  width: 18px;
```

```
  height: 18px;
  font-size: 12px;
  margin-left: auto;
}

.task-list {
  list-style: none;
  padding: 0;
  margin: 0;
}

.task-item {
  padding: var(--spacing-sm);
  border-bottom: 1px solid #eee;
  position: relative;
}

.task-item:last-child {
  border-bottom: none;
}

.task-priority {
  position: absolute;
  left: 0;
  top: 0;
  bottom: 0;
  width: 4px;
}

.priority-high {
  background-color: var(--light-purple);
}
```

```css
.priority-medium {
  background-color: var(--soft-yellow);
}

.priority-low {
  background-color: var(--soft-green);
}

.task-content {
  padding-left: var(--spacing-sm);
}

.task-name {
  font-weight: 500;
  margin-bottom: 4px;
}

.task-due {
  font-size: 12px;
  color: #666;
}

.overdue {
  color: var(--light-purple);
  font-weight: 500;
}

.counselor-list {
  list-style: none;
  padding: 0;
  margin: 0;
}
```

```css
.counselor-item {
  padding: var(--spacing-sm);
  border-bottom: 1px solid #eee;
  display: flex;
  align-items: center;
}

.counselor-item:last-child {
  border-bottom: none;
}

.counselor-status {
  width: 8px;
  height: 8px;
  border-radius: 50%;
  margin-right: var(--spacing-sm);
}

.status-active {
  background-color: var(--teal);
}

.status-away {
  background-color: var(--soft-yellow);
}

.status-offline {
  background-color: #ccc;
}

.counselor-name {
  flex: 1;
}
```

```css
.counselor-team {
  font-size: 12px;
  color: #666;
  padding: 2px 6px;
  background: #eee;
  border-radius: 10px;
}

.btn {
  border: none;
  padding: var(--spacing-sm) var(--spacing-md);
  border-radius: var(--border-radius-sm);
  font-weight: 500;
  cursor: pointer;
  transition: all 0.2s ease;
  width: 100%;
  text-align: center;
}

.btn-primary {
  background: var(--primary-gradient);
  color: white;
}

.btn-primary:hover {
  box-shadow: 0 4px 8px rgba(255, 120, 110, 0.3);
  transform: translateY(-2px);
}

.loading {
  text-align: center;
  padding: var(--spacing-md);
```

```
    color: #666;
  }
 </style>
</head>
<body>
 <div class="sidebar-container">
  <div class="sidebar-header">
    <h1>988 Lifeline Support</h1>
  </div>

  <div class="sidebar-content">
    <!-- User Summary -->
    <div class="card">
      <div class="card-header">
        <span>Team Lead Dashboard</span>
        <span id="currentDate">--/--/--</span>
      </div>
      <div class="card-content">
        <div class="metrics-grid">
          <div class="metric">
            <div class="metric-value" id="answerRate">--</div>
            <div class="metric-label">Answer Rate</div>
          </div>
          <div class="metric">
            <div class="metric-value" id="qualityScore">--</div>
            <div class="metric-label">Quality Score</div>
          </div>
          <div class="metric">
            <div class="metric-value" id="activeCounselors">--</div>
            <div class="metric-label">Active Counselors</div>
          </div>
          <div class="metric">
            <div class="metric-value" id="openTasks">--</div>
```

```
            <div class="metric-label">Open Tasks</div>
          </div>
        </div>
      </div>

      <!-- Quick Links -->
      <div class="card">
        <div class="card-header">
          <span>Quick Actions</span>
        </div>
        <div class="card-content">
          <ul class="quick-links">
            <li class="quick-link" onclick="openDashboard()">
              <span class="quick-link-icon">📊</span>
              <span>Open Dashboard</span>
            </li>
            <li class="quick-link" onclick="initializeShift()">
              <span class="quick-link-icon">⏱️</span>
              <span>Initialize Shift</span>
            </li>
            <li class="quick-link" onclick="trackTime()">
              <span class="quick-link-icon">⏰</span>
              <span>Track Time</span>
            </li>
            <li class="quick-link" onclick="viewAlerts()">
              <span class="quick-link-icon">🚨</span>
              <span>View Alerts</span>
              <span id="alertCount" class="alert-count" style="display:
none;">0</span>
            </li>
            <li class="quick-link" onclick="createTask()">
              <span class="quick-link-icon">✅</span>
```

```
            <span>Create Task</span>
          </li>
        </ul>
      </div>
    </div>

    <!-- My Tasks -->
    <div class="card">
      <div class="card-header">
        <span>My Tasks</span>
        <span id="taskCount">0</span>
      </div>
      <div class="card-content">
        <div id="tasksLoading" class="loading">Loading tasks...</div>
        <ul id="taskList" class="task-list" style="display: none;"></ul>
        <div id="noTasks" style="display: none; text-align: center; padding:
var(--spacing-md);">
          No tasks assigned to you
        </div>
      </div>
    </div>

    <!-- Active Counselors -->
    <div class="card">
      <div class="card-header">
        <span>Active Counselors</span>
        <span id="counselorCount">0</span>
      </div>
      <div class="card-content">
        <div id="counselorsLoading" class="loading">Loading
counselors...</div>
        <ul id="counselorList" class="counselor-list" style="display:
none;"></ul>
```

```
        <div id="noCounselors" style="display: none; text-align: center;
padding: var(--spacing-md);">
          No active counselors
        </div>
      </div>
    </div>
  </div>

  <div class="sidebar-footer">
    <button class="btn btn-primary" onclick="openTimeTracker()">Start Time
Tracking</button>
  </div>
 </div>
  <script>
  // Initialize sidebar
  document.addEventListener('DOMContentLoaded', function() {
    // Set current date
    const today = new Date();
    document.getElementById('currentDate').textContent =
today.toLocaleDateString();

    // Load metrics
    loadMetrics();

    // Load tasks
    loadTasks();

    // Load counselors
    loadCounselors();

    // Load alerts count
    loadAlertCount();
  });
```

```
  // Load metrics summary
  function loadMetrics() {
    google.script.run
      .withSuccessHandler(function(metrics) {
        document.getElementById('answerRate').textContent =
metrics.answerRate + '%';
        document.getElementById('qualityScore').textContent =
metrics.qualityScore;
        document.getElementById('activeCounselors').textContent =
metrics.activeCounselors;
        document.getElementById('openTasks').textContent = metrics.openTasks;
      })
      .withFailureHandler(handleError)
      .getDashboardMetrics();
  }

  // Load tasks assigned to current user
  function loadTasks() {
    document.getElementById('tasksLoading').style.display = 'block';
    document.getElementById('taskList').style.display = 'none';
    document.getElementById('noTasks').style.display = 'none';

    google.script.run
      .withSuccessHandler(function(tasks) {
        document.getElementById('tasksLoading').style.display = 'none';
        document.getElementById('taskCount').textContent = tasks.length;

        if (tasks.length === 0) {
          document.getElementById('noTasks').style.display = 'block';
          return;
        }
```

```
        const taskList = document.getElementById('taskList');
        taskList.innerHTML = '';

        tasks.forEach(function(task) {
          const li = document.createElement('li');
          li.className = 'task-item';

          // Format due date
          let dueDateText = 'No due date';
          let overdueClass = '';

          if (task.dueDate) {
            const dueDate = new Date(task.dueDate);
            dueDateText = 'Due: ' + dueDate.toLocaleDateString();

            // Check if overdue
            if (task.isOverdue) {
              overdueClass = 'overdue';
            }
          }

          li.innerHTML = `
            <div class="task-priority
priority-${task.priority.toLowerCase()}"></div>
            <div class="task-content">
              <div class="task-name">${task.task}</div>
              <div class="task-due ${overdueClass}">${dueDateText}</div>
            </div>
          `;

          taskList.appendChild(li);
        });
```

```
        document.getElementById('taskList').style.display = 'block';
      })
      .withFailureHandler(handleError)
      .getMyTasks();
  }

  // Load active counselors
  function loadCounselors() {
    document.getElementById('counselorsLoading').style.display = 'block';
    document.getElementById('counselorList').style.display = 'none';
    document.getElementById('noCounselors').style.display = 'none';

    google.script.run
      .withSuccessHandler(function(counselors) {
        document.getElementById('counselorsLoading').style.display = 'none';
        document.getElementById('counselorCount').textContent =
counselors.length;

        if (counselors.length === 0) {
          document.getElementById('noCounselors').style.display = 'block';
          return;
        }

        const counselorList = document.getElementById('counselorList');
        counselorList.innerHTML = '';

        counselors.forEach(function(counselor) {
          const li = document.createElement('li');
          li.className = 'counselor-item';

          let statusClass = 'status-offline';
          if (counselor.status === 'Active') {
            statusClass = 'status-active';
```

```
        } else if (counselor.status === 'Away') {
          statusClass = 'status-away';
        }

        li.innerHTML = `
          <div class="counselor-status ${statusClass}"></div>
          <div class="counselor-name">${counselor.name}</div>
          <div class="counselor-team">${counselor.team}</div>
        `;

        counselorList.appendChild(li);
      });

      document.getElementById('counselorList').style.display = 'block';
    })
    .withFailureHandler(handleError)
    .getActiveCounselors();
}

// Load alerts count
function loadAlertCount() {
  google.script.run
    .withSuccessHandler(function(count) {
      const alertCountElement = document.getElementById('alertCount');

      if (count > 0) {
        alertCountElement.textContent = count;
        alertCountElement.style.display = 'inline-flex';
      } else {
        alertCountElement.style.display = 'none';
      }
    })
    .withFailureHandler(handleError)
```

```
    .getActiveAlertsCount();
}

// Open dashboard
function openDashboard() {
  google.script.run.showDashboard();
}

// Initialize shift
function initializeShift() {
  google.script.run.showShiftInitialization();
}

// Track time
function trackTime() {
  google.script.run.showTimeTracker();
}

// View alerts
function viewAlerts() {
  google.script.run.showActiveAlerts();
}

// Create task
function createTask() {
  google.script.run.showTaskForm();
}

// Open time tracker
function openTimeTracker() {
  google.script.run.showTimeTracker();
}
```

```
    // Handle errors
    function handleError(error) {
      console.error('Error:', error);
    }
  </script>
</body>
</html>
<!DOCTYPE html>
<html>
<head>
  <base target="_top">
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link
href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500;700&fam
ily=Poppins:wght@500;600;700&display=swap" rel="stylesheet">
  <style>
    /* Variables for 988 Lifeline LGBTQIA+ Services color scheme */
    :root {
      /* Primary Colors */
      --trevor-orange: #FF786E;
      --deep-blue: #001A4E;
      --purple: #9A3499;
      --teal: #137F6A;

      /* Secondary Colors */
      --light-blue: #4F52DE;
      --soft-yellow: #FFAD8D;
      --lavender: #B3AE4A;
      --light-purple: #F54AC;

      /* Tertiary Colors */
      --soft-green: #BAE2CE;
```

```
    --pale-yellow: #FFF2DF;
    --light-pink: #FBCBBE;
    --soft-lavender: #D1CFCC;

    /* Gradients */
    --primary-gradient: linear-gradient(135deg, var(--trevor-orange),
var(--light-pink));
    --calm-gradient: linear-gradient(135deg, var(--light-blue),
var(--soft-green));
    --support-gradient: linear-gradient(45deg, var(--purple),
var(--light-purple));
    --background-gradient: linear-gradient(180deg, var(--pale-yellow),
#FFFFFF);

    /* Spacing */
    --spacing-xs: 4px;
    --spacing-sm: 8px;
    --spacing-md: 16px;
    --spacing-lg: 24px;
    --spacing-xl: 32px;

    /* Borders */
    --border-radius-sm: 8px;
    --border-radius-md: 12px;
    --border-radius-lg: 16px;
  }

  body {
    font-family: 'Roboto', sans-serif;
    margin: 0;
    padding: 0;
    background-color: var(--pale-yellow);
    color: var(--deep-blue);
```

```
}

.container {
  max-width: 600px;
  margin: 0 auto;
  padding: var(--spacing-md);
}

.header {
  margin-bottom: var(--spacing-lg);
  text-align: center;
}

h1, h2, h3, h4 {
  font-family: 'Poppins', sans-serif;
  color: var(--deep-blue);
}

.form-card {
  background: white;
  border-radius: var(--border-radius-md);
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.05);
  padding: var(--spacing-lg);
  position: relative;
}

.form-card::before {
  content: '';
  position: absolute;
  left: 0;
  top: 0;
  height: 100%;
  width: 3px;
```

```
  background: var(--light-blue);
}

.form-group {
  margin-bottom: var(--spacing-md);
}

label {
  display: block;
  font-weight: 500;
  margin-bottom: var(--spacing-xs);
}

input, select, textarea {
  width: 100%;
  padding: var(--spacing-sm);
  border: 1px solid #ddd;
  border-radius: var(--border-radius-sm);
  font-family: 'Roboto', sans-serif;
  font-size: 16px;
}

textarea {
  resize: vertical;
  min-height: 100px;
}

.required-field::after {
  content: " *";
  color: var(--trevor-orange);
}

.btn {
```

```
  border: none;
  padding: var(--spacing-md) var(--spacing-lg);
  border-radius: var(--border-radius-sm);
  font-weight: 500;
  cursor: pointer;
  transition: all 0.2s ease;
}

.btn-primary {
  background: var(--primary-gradient);
  color: white;
}

.btn-primary:hover {
  box-shadow: 0 4px 8px rgba(255, 120, 110, 0.3);
  transform: translateY(-2px);
}

.btn-secondary {
  background: white;
  color: var(--deep-blue);
  border: 1px solid #ddd;
}

.btn-secondary:hover {
  background: #f5f5f5;
}

.form-actions {
  display: flex;
  justify-content: space-between;
  margin-top: var(--spacing-lg);
}
```

```css
.success-message {
  background-color: var(--soft-green);
  color: var(--deep-blue);
  padding: var(--spacing-md);
  border-radius: var(--border-radius-sm);
  margin-bottom: var(--spacing-lg);
  text-align: center;
  display: none;
}

.error-message {
  background-color: var(--light-pink);
  color: var(--deep-blue);
  padding: var(--spacing-md);
  border-radius: var(--border-radius-sm);
  margin-bottom: var(--spacing-lg);
  text-align: center;
  display: none;
}

.priority-options {
  display: flex;
  gap: var(--spacing-sm);
  margin-top: var(--spacing-xs);
}

.priority-option {
  flex: 1;
  padding: var(--spacing-sm);
  border-radius: var(--border-radius-sm);
  text-align: center;
  cursor: pointer;
```

```
  transition: all 0.2s;
  border: 2px solid transparent;
}

.priority-option:hover {
  transform: translateY(-2px);
}

.priority-option.selected {
  border-color: var(--deep-blue);
}

.option-high {
  background-color: var(--light-pink);
}

.option-medium {
  background-color: var(--soft-yellow);
}

.option-low {
  background-color: var(--soft-green);
}

.help-text {
  font-size: 14px;
  color: #666;
  margin-top: var(--spacing-xs);
}

.form-row {
  display: flex;
  gap: var(--spacing-md);
```

```
    }

    .form-col {
      flex: 1;
    }

    @media (max-width: 600px) {
      .form-row {
        flex-direction: column;
        gap: 0;
      }
    }
  </style>
</head>
<body>
  <div class="container">
    <div class="header">
      <h1>Create Task</h1>
      <p>Add a new task to the 988 Lifeline LGBTQIA+ Crisis Support System</p>
    </div>

    <div id="successMessage" class="success-message">
      Task created successfully!
    </div>

    <div id="errorMessage" class="error-message">
      An error occurred. Please try again.
    </div>

    <div class="form-card">
      <form id="taskForm">
        <div class="form-group">
          <label for="taskTitle" class="required-field">Task Title</label>
```

```
          <input type="text" id="taskTitle" name="taskTitle" placeholder="Enter
a clear task title..." required>
        </div>

        <div class="form-group">
          <label for="taskDescription">Description</label>
          <textarea id="taskDescription" name="taskDescription"
placeholder="Describe the task in detail..."></textarea>
        </div>

        <div class="form-row">
          <div class="form-col">
            <div class="form-group">
              <label for="assignedTo" class="required-field">Assigned
To</label>
              <select id="assignedTo" name="assignedTo" required>
                <option value="">-- Select Assignee --</option>
                <!-- Team members will be populated here -->
              </select>
            </div>
          </div>

          <div class="form-col">
            <div class="form-group">
              <label for="dueDate">Due Date</label>
              <input type="date" id="dueDate" name="dueDate">
            </div>
          </div>
        </div>

        <div class="form-group">
          <label class="required-field">Priority</label>
          <div class="priority-options">
```

```
            <div class="priority-option option-high" data-priority="High"
onclick="selectPriority('High')">
              High
            </div>
            <div class="priority-option option-medium" data-priority="Medium"
onclick="selectPriority('Medium')">
              Medium
            </div>
            <div class="priority-option option-low" data-priority="Low"
onclick="selectPriority('Low')">
              Low
            </div>
          </div>
          <input type="hidden" id="taskPriority" name="taskPriority" value="">
        </div>

        <div class="form-group">
          <label for="category">Category</label>
          <select id="category" name="category">
            <option value="">-- Select Category --</option>
            <option value="Administrative">Administrative</option>
            <option value="Training">Training</option>
            <option value="LGBTQIA+ Resources">LGBTQIA+ Resources</option>
            <option value="System Development">System Development</option>
            <option value="Quality Improvement">Quality Improvement</option>
            <option value="Counselor Support">Counselor Support</option>
            <option value="Other">Other</option>
          </select>
        </div>

        <div class="form-actions">
          <button type="button" class="btn btn-secondary"
onclick="cancelForm()">Cancel</button>
```

```
        <button type="submit" class="btn btn-primary">Create Task</button>
      </div>
    </form>
  </div>
</div>
 <script>
  // Global variables
  let selectedPriority = null;

  // Initialize the form
  function initialize() {
    // Load team members for assignment dropdown
    loadTeamMembers();

    // Set up form submission
    document.getElementById('taskForm').addEventListener('submit',
submitForm);

    // Set default due date to 7 days from now
    const dueDate = new Date();
    dueDate.setDate(dueDate.getDate() + 7);
    document.getElementById('dueDate').value =
dueDate.toISOString().split('T')[0];
  }

  // Load team members for assignment dropdown
  function loadTeamMembers() {
    google.script.run
      .withSuccessHandler(function(members) {
        const select = document.getElementById('assignedTo');

        members.forEach(function(member) {
          const option = document.createElement('option');
```

```
          option.value = member.email;
          option.textContent = member.name;
          select.appendChild(option);
        });
      })
      .withFailureHandler(function(error) {
        console.error('Failed to load team members:', error);
      })
      .getTeamMembers();
}

// Select priority level
function selectPriority(priority) {
  selectedPriority = priority;

  // Update hidden input
  document.getElementById('taskPriority').value = priority;

  // Update UI
  document.querySelectorAll('.priority-option').forEach(function(option) {
    option.classList.remove('selected');
    if (option.getAttribute('data-priority') === priority) {
      option.classList.add('selected');
    }
  });
}

// Submit form
function submitForm(event) {
  event.preventDefault();

  // Hide messages
  document.getElementById('successMessage').style.display = 'none';
```

```
    document.getElementById('errorMessage').style.display = 'none';

    // Validate required fields
    const taskTitle = document.getElementById('taskTitle').value;
    const assignedTo = document.getElementById('assignedTo').value;

    if (!taskTitle) {
      document.getElementById('errorMessage').textContent = 'Please enter a
task title.';
      document.getElementById('errorMessage').style.display = 'block';
      return;
    }

    if (!assignedTo) {
      document.getElementById('errorMessage').textContent = 'Please select an
assignee.';
      document.getElementById('errorMessage').style.display = 'block';
      return;
    }

    if (!selectedPriority) {
      document.getElementById('errorMessage').textContent = 'Please select a
priority level.';
      document.getElementById('errorMessage').style.display = 'block';
      return;
    }

    // Get form data
    const taskData = {
      title: taskTitle,
      description: document.getElementById('taskDescription').value,
      assignedTo: assignedTo,
      dueDate: document.getElementById('dueDate').value,
```

```
      priority: selectedPriority,
      category: document.getElementById('category').value
    };

    // Submit to server
    google.script.run
      .withSuccessHandler(function(result) {
        if (result.success) {
          // Show success message
          document.getElementById('successMessage').textContent =
result.message || 'Task created successfully!';
          document.getElementById('successMessage').style.display = 'block';

          // Reset form
          document.getElementById('taskForm').reset();

          // Reset priority selection

document.querySelectorAll('.priority-option').forEach(function(option) {
            option.classList.remove('selected');
          });
          selectedPriority = null;

          // Set default due date again
          const dueDate = new Date();
          dueDate.setDate(dueDate.getDate() + 7);
          document.getElementById('dueDate').value =
dueDate.toISOString().split('T')[0];
        } else {
          // Show error message
          document.getElementById('errorMessage').textContent =
result.message || 'An error occurred while creating the task.';
          document.getElementById('errorMessage').style.display = 'block';
```

```
        }
      })
      .withFailureHandler(function(error) {
        // Show error message
        document.getElementById('errorMessage').textContent = 'Error creating
task: ' + error;
        document.getElementById('errorMessage').style.display = 'block';
      })
      .createTask(taskData);
  }

  // Cancel form
  function cancelForm() {
    google.script.host.close();
  }

  // Initialize when the page loads
  google.script.onload = function() {
    initialize();
  };
 </script>
</body>
</html>
/**
* TimeTracker.gs
*
* Service for tracking time spent on different activities
*/
const TimeTracker = {
 /**
  * Records an activity timestamp
  * @param {object} activityData - Activity data
  * @return {object} Result of the operation
```

```
 */
recordActivity: function(activityData) {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    let sheet = ss.getSheetByName('Activity Breakdown');

    // Create sheet if it doesn't exist
    if (!sheet) {
      sheet = ss.insertSheet('Activity Breakdown');
      sheet.appendRow([
        'Date',
        'User',
        'Activity Type',
        'Start Time',
        'End Time',
        'Duration (seconds)',
        'Duration (formatted)',
        'Notes'
      ]);
      sheet.getRange(1, 1, 1, 8)
        .setFontWeight('bold')
        .setBackground('#E0E0E0');
    }

    // Validate required fields
    if (!activityData.activityType) {
      return {
        success: false,
        message: 'Activity type is required'
      };
    }

    // Calculate duration in seconds
```

```
    let durationSeconds = 0;
    if (activityData.startTime && activityData.endTime) {
      const startTime = new Date(activityData.startTime);
      const endTime = new Date(activityData.endTime);
      durationSeconds = Math.round((endTime - startTime) / 1000);
    } else if (activityData.durationSeconds) {
      durationSeconds = activityData.durationSeconds;
    }

    // Format duration as HH:MM:SS
    const hours = Math.floor(durationSeconds / 3600);
    const minutes = Math.floor((durationSeconds % 3600) / 60);
    const seconds = durationSeconds % 60;
    const formattedDuration = `${hours.toString().padStart(2,
'0')}:${minutes.toString().padStart(2, '0')}:${seconds.toString().padStart(2,
'0')}`;

    // Get user info
    const user = Session.getActiveUser().getEmail();

    // Prepare row data
    const rowData = [
      new Date(activityData.date) || new Date(),
      user,
      activityData.activityType,
      activityData.startTime ? new Date(activityData.startTime) : '',
      activityData.endTime ? new Date(activityData.endTime) : '',
      durationSeconds,
      formattedDuration,
      activityData.notes || ''
    ];

    // Add to sheet
```

```
    sheet.appendRow(rowData);

    // Update time log with new activity
    this.updateTimeLog(activityData.date, activityData.activityType,
durationSeconds);

    return {
      success: true,
      message: 'Activity recorded successfully'
    };
  } catch (error) {
    logError('recordActivity', error);
    return {
      success: false,
      message: 'Error recording activity: ' + error.message
    };
  }
},
 /**
 * Updates the time log with a new activity
 * @param {Date|string} date - Activity date
 * @param {string} activityType - Type of activity
 * @param {number} durationSeconds - Duration in seconds
 * @private
 */
updateTimeLog: function(date, activityType, durationSeconds) {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    let sheet = ss.getSheetByName('Time Logs');

    // Create sheet if it doesn't exist
    if (!sheet) {
      sheet = ss.insertSheet('Time Logs');
```

```
      sheet.appendRow([
        'Date',
        'Primary Shift Time',
        'Admin Time',
        'Meeting Time',
        '1:1 Time',
        'Other Time',
        'Total Time',
        'Activity Notes',
        'Tasks Accomplished',
        'Summary Notes',
        'User',
        'Timestamp'
      ]);
      sheet.getRange(1, 1, 1, 12)
        .setFontWeight('bold')
        .setBackground('#E0E0E0');
    }

    // Convert duration to minutes
    const durationMinutes = durationSeconds / 60;

    // Format date to YYYY-MM-DD for consistent comparison
    const activityDate = date instanceof Date ? date : new Date(date);
    const dateString = Utilities.formatDate(activityDate,
Session.getScriptTimeZone(), 'yyyy-MM-dd');

    // Get user info
    const user = Session.getActiveUser().getEmail();

    // Look for an existing log entry for this date and user
    const data = sheet.getDataRange().getValues();
    let rowIndex = -1;
```

```
    for (let i = 1; i < data.length; i++) {
      const rowDate = data[i][0];
      const rowUser = data[i][10]; // User column

      if (rowDate && rowUser === user) {
        // Format date for comparison
        const rowDateString = Utilities.formatDate(rowDate,
Session.getScriptTimeZone(), 'yyyy-MM-dd');

        if (rowDateString === dateString) {
          rowIndex = i + 1; // +1 because sheet rows are 1-indexed
          break;
        }
      }
    }

    // If no existing entry, create a new one
    if (rowIndex === -1) {
      sheet.appendRow([
        activityDate,
        0, // Primary Shift Time
        0, // Admin Time
        0, // Meeting Time
        0, // 1:1 Time
        0, // Other Time
        0, // Total Time
        '', // Activity Notes
        '', // Tasks Accomplished
        '', // Summary Notes
        user, // User
        new Date() // Timestamp
      ]);
```

```
      rowIndex = sheet.getLastRow();
    }

    // Get current values
    const currentValues = sheet.getRange(rowIndex, 2, 1, 6).getValues()[0];

    // Update the appropriate time column based on activity type
    let columnIndex;
    switch (activityType) {
      case 'Primary Shift':
        columnIndex = 2; // Column B
        break;
      case 'Administrative':
        columnIndex = 3; // Column C
        break;
      case 'Meetings':
        columnIndex = 4; // Column D
        break;
      case '1:1 Coaching':
        columnIndex = 5; // Column E
        break;
      default:
        columnIndex = 6; // Column F (Other)
    }

    // Get current value and add new duration
    const currentValue = currentValues[columnIndex - 2];
    const newValue = currentValue + durationMinutes;
    sheet.getRange(rowIndex, columnIndex).setValue(newValue);

    // Update total time (column G)
    const totalMinutes = currentValues[0] + currentValues[1] +
currentValues[2] + currentValues[3] + currentValues[4] + durationMinutes;
```

```
    sheet.getRange(rowIndex, 7).setValue(totalMinutes);

    // Update timestamp
    sheet.getRange(rowIndex, 12).setValue(new Date());

    // Update activity summary for the day
    this.updateActivitySummary();
  } catch (error) {
    logError('updateTimeLog', error);
  }
},
 /**
 * Gets time log data for the current user
 * @param {object} options - Options for filtering time logs
 * @return {object} Time log data
 */
getTimeLog: function(options = {}) {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    const sheet = ss.getSheetByName('Time Logs');

    if (!sheet) {
      return {
        success: false,
        message: 'Time Logs sheet not found'
      };
    }

    // Get all data
    const data = sheet.getDataRange().getValues();

    // Skip header row
    if (data.length <= 1) {
```

```
      return {
        success: false,
        message: 'No time log data found'
      };
    }

    // Get user info
    const user = options.user || Session.getActiveUser().getEmail();

    // Filter by date if provided
    let filteredData = data.slice(1).filter(row => row[10] === user); // 
Filter by user

    if (options.date) {
      const targetDate = options.date instanceof Date ? options.date : new 
Date(options.date);
      const targetDateString = Utilities.formatDate(targetDate, 
Session.getScriptTimeZone(), 'yyyy-MM-dd');

      filteredData = filteredData.filter(row => {
        const rowDate = row[0];
        if (!rowDate) return false;

        const rowDateString = Utilities.formatDate(rowDate, 
Session.getScriptTimeZone(), 'yyyy-MM-dd');
        return rowDateString === targetDateString;
      });
    }

    // Map to an array of time log objects
    const timeLogs = filteredData.map(row => ({
      date: row[0],
      primaryShiftTime: row[1],
```

```
      adminTime: row[2],
      meetingTime: row[3],
      oneOnOneTime: row[4],
      otherTime: row[5],
      totalTime: row[6],
      activityNotes: row[7],
      tasksAccomplished: row[8],
      summaryNotes: row[9],
      user: row[10],
      timestamp: row[11]
    }));

    // For today's log, if requested
    if (options.today) {
      const today = new Date();
      const todayString = Utilities.formatDate(today,
Session.getScriptTimeZone(), 'yyyy-MM-dd');

      const todayLog = timeLogs.find(log => {
        const logDate = log.date;
        if (!logDate) return false;

        const logDateString = Utilities.formatDate(logDate,
Session.getScriptTimeZone(), 'yyyy-MM-dd');
        return logDateString === todayString;
      });

      if (todayLog) {
        return {
          success: true,
          found: true,
          log: todayLog,
          times: {
```

```
            'Primary Shift': todayLog.primaryShiftTime,
            'Administrative': todayLog.adminTime,
            'Meetings': todayLog.meetingTime,
            '1:1 Coaching': todayLog.oneOnOneTime,
            'Other': todayLog.otherTime
          }
        };
      } else {
        return {
          success: true,
          found: false,
          message: 'No time log found for today',
          times: {
            'Primary Shift': 0,
            'Administrative': 0,
            'Meetings': 0,
            '1:1 Coaching': 0,
            'Other': 0
          }
        };
      }
    }

    return {
      success: true,
      logs: timeLogs
    };
  } catch (error) {
    logError('getTimeLog', error);
    return {
      success: false,
      message: 'Error retrieving time log: ' + error.message
    };
```

```
  }
},
 /**
 * Gets activity breakdown data for the current user
 * @param {object} options - Options for filtering activities
 * @return {object} Activity breakdown data
 */
getActivityBreakdown: function(options = {}) {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    const sheet = ss.getSheetByName('Activity Breakdown');

    if (!sheet) {
      return {
        success: false,
        message: 'Activity Breakdown sheet not found'
      };
    }

    // Get all data
    const data = sheet.getDataRange().getValues();

    // Skip header row
    if (data.length <= 1) {
      return {
        success: false,
        message: 'No activity data found'
      };
    }

    // Get column indices
    const headers = data[0];
    const dateIndex = headers.indexOf('Date');
```

```
    const userIndex = headers.indexOf('User');
    const activityTypeIndex = headers.indexOf('Activity Type');
    const startTimeIndex = headers.indexOf('Start Time');
    const endTimeIndex = headers.indexOf('End Time');
    const durationSecondsIndex = headers.indexOf('Duration (seconds)');
    const durationFormattedIndex = headers.indexOf('Duration (formatted)');
    const notesIndex = headers.indexOf('Notes');

    // Check if required columns exist
    if (dateIndex === -1 || userIndex === -1 || activityTypeIndex === -1) {
      return {
        success: false,
        message: 'Required columns not found in Activity Breakdown sheet'
      };
    }

    // Get user info
    const user = options.user || Session.getActiveUser().getEmail();

    // Filter by user
    let filteredData = data.slice(1).filter(row => row[userIndex] === user);

    // Filter by date if provided
    if (options.date) {
      const targetDate = options.date instanceof Date ? options.date : new
Date(options.date);
      const targetDateString = Utilities.formatDate(targetDate,
Session.getScriptTimeZone(), 'yyyy-MM-dd');

      filteredData = filteredData.filter(row => {
        const rowDate = row[dateIndex];
        if (!rowDate) return false;
```

```
        const rowDateString = Utilities.formatDate(rowDate,
Session.getScriptTimeZone(), 'yyyy-MM-dd');
        return rowDateString === targetDateString;
      });
    }

    // Map to an array of activity objects
    const activities = filteredData.map(row => ({
      date: row[dateIndex],
      user: row[userIndex],
      activityType: row[activityTypeIndex],
      startTime: row[startTimeIndex],
      endTime: row[endTimeIndex],
      durationSeconds: row[durationSecondsIndex],
      durationFormatted: row[durationFormattedIndex],
      notes: row[notesIndex]
    }));

    // Sort by start time (newest first)
    activities.sort((a, b) => {
      if (!a.startTime) return 1;
      if (!b.startTime) return -1;
      return b.startTime - a.startTime;
    });

    // For recent activities, if requested
    if (options.recent) {
      const recentCount = options.count || 10;
      return {
        success: true,
        activities: activities.slice(0, recentCount)
      };
    }
```

```
    // Create a summary by activity type
    const summary = {};

    activities.forEach(activity => {
      if (!summary[activity.activityType]) {
        summary[activity.activityType] = 0;
      }
      summary[activity.activityType] += activity.durationSeconds;
    });

    // Convert seconds to hours:minutes:seconds in summary
    Object.keys(summary).forEach(key => {
      const totalSeconds = summary[key];
      const hours = Math.floor(totalSeconds / 3600);
      const minutes = Math.floor((totalSeconds % 3600) / 60);
      const seconds = totalSeconds % 60;
      summary[key] = {
        seconds: totalSeconds,
        formatted: `${hours.toString().padStart(2,
'0')}:${minutes.toString().padStart(2, '0')}:${seconds.toString().padStart(2,
'0')}`
      };
    });

    return {
      success: true,
      activities: activities,
      summary: summary
    };
  } catch (error) {
    logError('getActivityBreakdown', error);
    return {
```

```
      success: false,
      message: 'Error retrieving activity breakdown: ' + error.message
    };
  }
},
 /**
 * Updates the activity summary for the week
 * @private
 */
updateActivitySummary: function() {
  try {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    let sheet = ss.getSheetByName('Time Summary');

    // Create sheet if it doesn't exist
    if (!sheet) {
      sheet = ss.insertSheet('Time Summary');
      sheet.appendRow([
        'Week',
        'User',
        'Primary Shift Hours',
        'Admin Hours',
        'Meeting Hours',
        '1:1 Hours',
        'Other Hours',
        'Total Hours',
        'Tasks Count',
        'Last Updated'
      ]);
      sheet.getRange(1, 1, 1, 10)
        .setFontWeight('bold')
        .setBackground('#E0E0E0');
    }
```

```
    // Get current week
    const today = new Date();
    const weekStart = new Date(today);
    weekStart.setDate(today.getDate() - today.getDay()); // Start of week
(Sunday)
    weekStart.setHours(0, 0, 0, 0);

    const weekEnd = new Date(weekStart);
    weekEnd.setDate(weekStart.getDate() + 6); // End of week (Saturday)
    weekEnd.setHours(23, 59, 59, 999);

    const weekString = `${Utilities.formatDate(weekStart,
Session.getScriptTimeZone(), 'yyyy-MM-dd')} to ${Utilities.formatDate(weekEnd,
Session.getScriptTimeZone(), 'yyyy-MM-dd')}`;

    // Get user info
    const user = Session.getActiveUser().getEmail();

    // Get time logs for the week
    const timeLogSheet = ss.getSheetByName('Time Logs');
    if (!timeLogSheet) {
      return; // No time logs sheet, nothing to summarize
    }

    const timeLogData = timeLogSheet.getDataRange().getValues();
    if (timeLogData.length <= 1) {
      return; // No time log data, nothing to summarize
    }

    // Filter logs for this user and week
    const weekLogs = timeLogData.slice(1).filter(row => {
      if (row[10] !== user) return false; // Not this user
```

```
      const logDate = row[0];
      if (!logDate) return false;

      return logDate >= weekStart && logDate <= weekEnd;
    });

    // Sum up times
    let primaryShiftMinutes = 0;
    let adminMinutes = 0;
    let meetingMinutes = 0;
    let oneOnOneMinutes = 0;
    let otherMinutes = 0;

    weekLogs.forEach(log => {
      primaryShiftMinutes += log[1] || 0;
      adminMinutes += log[2] || 0;
      meetingMinutes += log[3] || 0;
      oneOnOneMinutes += log[4] || 0;
      otherMinutes += log[5] || 0;
    });

    // Calculate total minutes
    const totalMinutes = primaryShiftMinutes + adminMinutes + meetingMinutes
+ oneOnOneMinutes + otherMinutes;

    // Convert to hours
    const primaryShiftHours = primaryShiftMinutes / 60;
    const adminHours = adminMinutes / 60;
    const meetingHours = meetingMinutes / 60;
    const oneOnOneHours = oneOnOneMinutes / 60;
    const otherHours = otherMinutes / 60;
    const totalHours = totalMinutes / 60;
```

```
// Count tasks completed this week
const tasksSheet = ss.getSheetByName(CRISIS_SUPPORT_CONFIG.SHEETS.TASKS);
let tasksCount = 0;

if (tasksSheet) {
  const tasksData = tasksSheet.getDataRange().getValues();
  if (tasksData.length > 1) {
    const headers = tasksData[0];
    const completedDateIndex = headers.indexOf('Completed Date');
    const completedByIndex = headers.indexOf('Completed By');

    if (completedDateIndex !== -1 && completedByIndex !== -1) {
      tasksCount = tasksData.slice(1).filter(row => {
        const completedDate = row[completedDateIndex];
        const completedBy = row[completedByIndex];

        if (!completedDate || completedBy !== user) return false;

        return completedDate >= weekStart && completedDate <= weekEnd;
      }).length;
    }
  }
}

// Check if summary already exists for this week and user
const summaryData = sheet.getDataRange().getValues();
let rowIndex = -1;

for (let i = 1; i < summaryData.length; i++) {
  if (summaryData[i][0] === weekString && summaryData[i][1] === user) {
    rowIndex = i + 1; // +1 because sheet rows are 1-indexed
    break;
```

```
      }
    }

    // Update or create the summary row
    if (rowIndex !== -1) {
      // Update existing row
      sheet.getRange(rowIndex, 3, 1, 8).setValues([[
        primaryShiftHours.toFixed(2),
        adminHours.toFixed(2),
        meetingHours.toFixed(2),
        oneOnOneHours.toFixed(2),
        otherHours.toFixed(2),
        totalHours.toFixed(2),
        tasksCount,
        new Date()
      ]]);
    } else {
      // Create new row
      sheet.appendRow([
        weekString,
        user,
        primaryShiftHours.toFixed(2),
        adminHours.toFixed(2),
        meetingHours.toFixed(2),
        oneOnOneHours.toFixed(2),
        otherHours.toFixed(2),
        totalHours.toFixed(2),
        tasksCount,
        new Date()
      ]);
    }
  } catch (error) {
    logError('updateActivitySummary', error);
```

```
   }
 }
};

// Expose functions to UI
function recordActivity(activityData) {
 return TimeTracker.recordActivity(activityData);
}

function getTimeLog(options) {
 return TimeTracker.getTimeLog(options);
}

function getActivityBreakdown(options) {
 return TimeTracker.getActivityBreakdown(options);
}

function getTodaysTimeLog() {
 return TimeTracker.getTimeLog({ today: true });
}

function getRecentActivities() {
 return TimeTracker.getActivityBreakdown({ recent: true, count: 10
}).activities || [];
}
<!DOCTYPE html>
<html>
<head>
 <base target="_top">
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
 <link
href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500;700&fam
ily=Poppins:wght@500;600;700&display=swap" rel="stylesheet">
 <style>
   :root {
     /* Primary Colors */
     --trevor-orange: #FF786E;
     --deep-blue: #001A4E;
     --purple: #9A3499;
     --teal: #137F6A;

     /* Secondary Colors */
     --light-blue: #4F52DE;
     --soft-yellow: #FFAD8D;
     --light-purple: #F54AC;

     /* Tertiary Colors */
     --soft-green: #BAE2CE;
     --pale-yellow: #FFF2DF;
     --light-pink: #FBCBBE;

     /* Gradients */
     --primary-gradient: linear-gradient(135deg, var(--trevor-orange),
var(--light-pink));
     --background-gradient: linear-gradient(180deg, var(--pale-yellow),
#FFFFFF);

     /* Spacing */
     --spacing-xs: 4px;
     --spacing-sm: 8px;
     --spacing-md: 16px;
     --spacing-lg: 24px;
     --spacing-xl: 32px;
```

```
  /* Borders */
  --border-radius-sm: 8px;
  --border-radius-md: 12px;
  --border-radius-lg: 16px;
}

body {
  font-family: 'Roboto', sans-serif;
  background: var(--background-gradient);
  color: var(--deep-blue);
  margin: 0;
  padding: 0;
  min-height: 100vh;
}

.container {
  max-width: 800px;
  margin: 0 auto;
  padding: var(--spacing-lg);
}

h1, h2, h3 {
  font-family: 'Poppins', sans-serif;
  color: var(--deep-blue);
}

.header {
  margin-bottom: var(--spacing-xl);
  position: relative;
  padding-bottom: var(--spacing-md);
  border-bottom: 1px solid rgba(0, 0, 0, 0.1);
}
```

```css
.header h1 {
  margin-bottom: var(--spacing-xs);
}

.header p {
  color: #666;
  margin-top: 0;
}

.counselor-selector {
  margin-bottom: var(--spacing-xl);
}

.counselor-card {
  background: white;
  border-radius: var(--border-radius-md);
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.05);
  padding: var(--spacing-lg);
  margin-bottom: var(--spacing-lg);
  position: relative;
}

.counselor-card::before {
  content: '';
  position: absolute;
  left: 0;
  top: 0;
  height: 100%;
  width: 3px;
  background: var(--trevor-orange);
}
```

```css
.counselor-info {
  display: flex;
  align-items: center;
  margin-bottom: var(--spacing-md);
}

.counselor-avatar {
  width: 60px;
  height: 60px;
  background-color: var(--light-pink);
  border-radius: 50%;
  display: flex;
  align-items: center;
  justify-content: center;
  font-size: 24px;
  font-weight: 500;
  color: var(--trevor-orange);
  margin-right: var(--spacing-md);
}

.counselor-details h2 {
  margin: 0 0 var(--spacing-xs) 0;
}

.status-badge {
  display: inline-block;
  padding: 4px 8px;
  border-radius: 50px;
  font-size: 12px;
  font-weight: 500;
}

.status-active {
```

```
  background-color: rgba(19, 127, 106, 0.1);
  color: var(--teal);
}

.status-training {
  background-color: rgba(79, 82, 222, 0.1);
  color: var(--light-blue);
}

.status-pto {
  background-color: rgba(255, 173, 141, 0.1);
  color: var(--soft-yellow);
}

.status-loa {
  background-color: rgba(155, 52, 153, 0.1);
  color: var(--purple);
}

.status-inactive {
  background-color: rgba(245, 74, 12, 0.1);
  color: var(--light-purple);
}

.status-disappeared {
  background-color: rgba(245, 74, 12, 0.2);
  color: var(--light-purple);
}

.status-overtime {
  background-color: rgba(19, 127, 106, 0.2);
  color: var(--teal);
}
```

```css
.status-leaving-early {
  background-color: rgba(255, 173, 141, 0.2);
  color: var(--soft-yellow);
}

.counselor-meta {
  display: flex;
  gap: var(--spacing-lg);
  font-size: 14px;
}

.meta-item {
  margin-bottom: var(--spacing-xs);
}

.meta-label {
  color: #666;
}

.card {
  background: white;
  border-radius: var(--border-radius-md);
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.05);
  overflow: hidden;
  position: relative;
  margin-bottom: var(--spacing-xl);
}

.card::before {
  content: '';
  position: absolute;
  left: 0;
```

```
  top: 0;
  height: 100%;
  width: 3px;
  background: var(--trevor-orange);
}

.card-content {
  padding: var(--spacing-lg);
}

.form-group {
  margin-bottom: var(--spacing-lg);
}

.form-row {
  display: flex;
  flex-wrap: wrap;
  margin: 0 -10px;
}

.form-col {
  flex: 1;
  padding: 0 10px;
  min-width: 200px;
}

label {
  display: block;
  margin-bottom: var(--spacing-xs);
  font-weight: 500;
}

input, select, textarea {
```

```
  width: 100%;
  padding: 10px 12px;
  border: 1px solid #ddd;
  border-radius: var(--border-radius-sm);
  font-family: 'Roboto', sans-serif;
  font-size: 14px;
  transition: all 0.2s ease;
}

input:focus, select:focus, textarea:focus {
  outline: none;
  border-color: var(--trevor-orange);
  box-shadow: 0 0 0 3px rgba(255, 120, 110, 0.2);
}

textarea {
  min-height: 120px;
  resize: vertical;
}

.required::after {
  content: '*';
  color: var(--trevor-orange);
  margin-left: 2px;
}

.help-text {
  font-size: 12px;
  color: #666;
  margin-top: var(--spacing-xs);
}

.status-option {
```

```
  padding: var(--spacing-md);
  border: 2px solid #eee;
  border-radius: var(--border-radius-sm);
  margin-bottom: var(--spacing-md);
  cursor: pointer;
  transition: all 0.2s ease;
}

.status-option:hover {
  border-color: #ddd;
  background-color: #f9f9f9;
}

.status-option.selected {
  border-color: var(--trevor-orange);
  background-color: rgba(255, 120, 110, 0.05);
}

.status-option-header {
  display: flex;
  align-items: center;
  margin-bottom: var(--spacing-xs);
}

.status-radio {
  margin-right: var(--spacing-sm);
}

.status-title {
  font-weight: 500;
  flex-grow: 1;
}
```

```css
.status-description {
  font-size: 14px;
  color: #666;
}

.additional-fields {
  margin-top: var(--spacing-md);
  padding-top: var(--spacing-md);
  border-top: 1px solid #eee;
  display: none;
}

.btn {
  border: none;
  padding: var(--spacing-sm) var(--spacing-lg);
  border-radius: var(--border-radius-sm);
  font-weight: 500;
  cursor: pointer;
  transition: all 0.2s ease;
  font-family: 'Poppins', sans-serif;
}

.btn-primary {
  background: var(--primary-gradient);
  color: white;
}

.btn-primary:hover {
  box-shadow: 0 4px 8px rgba(255, 120, 110, 0.3);
  transform: translateY(-2px);
}

.btn-secondary {
```

```
  background: white;
  color: var(--deep-blue);
  border: 1px solid #ddd;
}

.btn-secondary:hover {
  background: #f5f5f5;
}

.form-actions {
  display: flex;
  justify-content: space-between;
  margin-top: var(--spacing-xl);
}

.notification {
  padding: var(--spacing-md);
  border-radius: var(--border-radius-sm);
  margin-bottom: var(--spacing-lg);
  display: none;
}

.notification.success {
  background-color: rgba(19, 127, 106, 0.1);
  color: var(--teal);
  border-left: 3px solid var(--teal);
}

.notification.error {
  background-color: rgba(245, 74, 12, 0.1);
  color: var(--light-purple);
  border-left: 3px solid var(--light-purple);
}
```

```
    .field-error {
      color: var(--light-purple);
      font-size: 12px;
      margin-top: var(--spacing-xs);
      display: none;
    }

    @media (max-width: 600px) {
      .form-col {
        flex: 100%;
        margin-bottom: var(--spacing-md);
      }

      .form-actions {
        flex-direction: column;
      }

      .form-actions button {
        margin-bottom: var(--spacing-sm);
      }

      .counselor-meta {
        flex-direction: column;
        gap: var(--spacing-sm);
      }
    }
  </style>
</head>
<body>
  <div class="container">
    <div class="header">
      <h1>Update Counselor Status</h1>
```

```
    <p>Update the status of team members for the 988 Lifeline LGBTQIA+
Services</p>
  </div>

  <div id="notification" class="notification"></div>

  <div class="counselor-selector">
    <label for="counselorSelect">Select Counselor:</label>
    <select id="counselorSelect" onchange="loadCounselorDetails()">
      <option value="">Choose a counselor...</option>
      <!-- Options will be populated dynamically -->
    </select>
  </div>

  <div id="counselorCard" class="counselor-card" style="display: none;">
    <div class="counselor-info">
      <div class="counselor-avatar" id="counselorInitials">JD</div>
      <div class="counselor-details">
        <h2 id="counselorName">John Doe</h2>
        <span class="status-badge status-active"
id="currentStatus">Active</span>
      </div>
    </div>

    <div class="counselor-meta">
      <div>
        <div class="meta-item">
          <span class="meta-label">Team:</span>
          <span id="counselorTeam">Digital</span>
        </div>
        <div class="meta-item">
          <span class="meta-label">Email:</span>
          <span id="counselorEmail">john.doe@example.com</span>
```

```
        </div>
      </div>
      <div>
        <div class="meta-item">
          <span class="meta-label">Phone:</span>
          <span id="counselorPhone">(555) 123-4567</span>
        </div>
        <div class="meta-item">
          <span class="meta-label">Start Date:</span>
          <span id="counselorStartDate">Jan 15, 2023</span>
        </div>
      </div>
    </div>
  </div>

  <div class="card" id="statusForm" style="display: none;">
    <div class="card-content">
      <form id="updateStatusForm">
        <h3>Select New Status</h3>

        <div class="status-options">
          <div class="status-option" onclick="selectStatus(this, 'Active')">
            <div class="status-option-header">
              <input type="radio" name="status" value="Active"
class="status-radio">
              <div class="status-title">Active</div>
              <span class="status-badge status-active">Active</span>
            </div>
            <div class="status-description">Counselor is actively working and
available for shifts.</div>
          </div>

          <div class="status-option" onclick="selectStatus(this, 'PTO')">
```

```
            <div class="status-option-header">
              <input type="radio" name="status" value="PTO"
class="status-radio">
              <div class="status-title">Planned Time Off (PTO)</div>
              <span class="status-badge status-pto">PTO</span>
            </div>
            <div class="status-description">Counselor is on approved,
scheduled time off.</div>
            <div class="additional-fields" id="ptoFields">
              <div class="form-row">
                <div class="form-col">
                  <div class="form-group">
                    <label for="ptoStartDate">Start Date</label>
                    <input type="date" id="ptoStartDate" name="ptoStartDate">
                  </div>
                </div>
                <div class="form-col">
                  <div class="form-group">
                    <label for="ptoEndDate">End Date</label>
                    <input type="date" id="ptoEndDate" name="ptoEndDate">
                  </div>
                </div>
              </div>
            </div>

          <div class="status-option" onclick="selectStatus(this, 'UTO')">
            <div class="status-option-header">
              <input type="radio" name="status" value="UTO"
class="status-radio">
              <div class="status-title">Unplanned Time Off (UTO)</div>
              <span class="status-badge status-loa">UTO</span>
            </div>
```

```
            <div class="status-description">Counselor is absent without prior
scheduling.</div>
            <div class="additional-fields" id="utoFields">
              <div class="form-group">
                <label for="utoReason">Reason (if known)</label>
                <input type="text" id="utoReason" name="utoReason">
              </div>
              <div class="form-row">
                <div class="form-col">
                  <div class="form-group">
                    <label for="utoDate">Date</label>
                    <input type="date" id="utoDate" name="utoDate">
                  </div>
                </div>
                <div class="form-col">
                  <div class="form-group">
                    <label for="utoShift">Shift</label>
                    <select id="utoShift" name="utoShift">
                      <option value="Morning">Morning</option>
                      <option value="Afternoon">Afternoon</option>
                      <option value="Evening">Evening</option>
                      <option value="Overnight">Overnight</option>
                      <option value="Full Day">Full Day</option>
                    </select>
                  </div>
                </div>
              </div>
            </div>

            <div class="status-option" onclick="selectStatus(this,
'Overtime')">
              <div class="status-option-header">
```

```
                <input type="radio" name="status" value="Overtime"
class="status-radio">
                <div class="status-title">Overtime</div>
                <span class="status-badge status-overtime">Overtime</span>
              </div>
              <div class="status-description">Counselor is working beyond
scheduled hours.</div>
              <div class="additional-fields" id="overtimeFields">
                <div class="form-row">
                  <div class="form-col">
                    <div class="form-group">
                      <label for="overtimeDate">Date</label>
                      <input type="date" id="overtimeDate" name="overtimeDate">
                    </div>
                  </div>
                  <div class="form-col">
                    <div class="form-group">
                      <label for="overtimeHours">Hours</label>
                      <input type="number" id="overtimeHours"
name="overtimeHours" min="0.5" max="24" step="0.5">
                    </div>
                  </div>
                </div>
                <div class="form-group">
                  <label for="overtimeReason">Reason</label>
                  <input type="text" id="overtimeReason" name="overtimeReason">
                </div>
              </div>
            </div>

            <div class="status-option" onclick="selectStatus(this,
'Disappeared')">
              <div class="status-option-header">
```

```
                <input type="radio" name="status" value="Disappeared"
class="status-radio">
                <div class="status-title">Disappeared from Shift</div>
                <span class="status-badge
status-disappeared">Disappeared</span>
              </div>
              <div class="status-description">Counselor was on shift but is no
longer responsive.</div>
              <div class="additional-fields" id="disappearedFields">
                <div class="form-group">
                  <label for="disappearedTime">Last Seen Time</label>
                  <input type="time" id="disappearedTime"
name="disappearedTime">
                </div>
                <div class="form-group">
                  <label for="disappearedActions">Actions Taken</label>
                  <textarea id="disappearedActions" name="disappearedActions"
placeholder="Describe actions taken to locate the counselor..."></textarea>
                </div>
              </div>
            </div>

            <div class="status-option" onclick="selectStatus(this,
'LeavingEarly')">
              <div class="status-option-header">
                <input type="radio" name="status" value="LeavingEarly"
class="status-radio">
                <div class="status-title">Leaving Early</div>
                <span class="status-badge status-leaving-early">Leaving
Early</span>
              </div>
              <div class="status-description">Counselor needs to leave before
scheduled end of shift.</div>
```

```
            <div class="additional-fields" id="leavingEarlyFields">
              <div class="form-row">
                <div class="form-col">
                  <div class="form-group">
                    <label for="leavingEarlyTime">Departure Time</label>
                    <input type="time" id="leavingEarlyTime"
name="leavingEarlyTime">
                  </div>
                </div>
                <div class="form-col">
                  <div class="form-group">
                    <label for="scheduledEndTime">Scheduled End Time</label>
                    <input type="time" id="scheduledEndTime"
name="scheduledEndTime">
                  </div>
                </div>
              </div>
              <div class="form-group">
                <label for="leavingEarlyReason">Reason</label>
                <input type="text" id="leavingEarlyReason"
name="leavingEarlyReason">
              </div>
            </div>
          </div>

          <div class="status-option" onclick="selectStatus(this, 'LOA')">
            <div class="status-option-header">
              <input type="radio" name="status" value="LOA"
class="status-radio">
              <div class="status-title">Leave of Absence (LOA)</div>
              <span class="status-badge status-loa">LOA</span>
            </div>
```

```
            <div class="status-description">Counselor is on an extended leave
of absence.</div>
            <div class="additional-fields" id="loaFields">
              <div class="form-row">
                <div class="form-col">
                  <div class="form-group">
                    <label for="loaStartDate">Start Date</label>
                    <input type="date" id="loaStartDate" name="loaStartDate">
                  </div>
                </div>
                <div class="form-col">
                  <div class="form-group">
                    <label for="loaEndDate">Expected Return Date</label>
                    <input type="date" id="loaEndDate" name="loaEndDate">
                  </div>
                </div>
              </div>
              <div class="form-group">
                <label for="loaType">Type of Leave</label>
                <select id="loaType" name="loaType">
                  <option value="Medical">Medical</option>
                  <option value="Family">Family</option>
                  <option value="Personal">Personal</option>
                  <option value="Educational">Educational</option>
                  <option value="Other">Other</option>
                </select>
              </div>
            </div>

            <div class="status-option" onclick="selectStatus(this,
'Inactive')">
              <div class="status-option-header">
```

```
              <input type="radio" name="status" value="Inactive"
class="status-radio">
              <div class="status-title">Inactive</div>
              <span class="status-badge status-inactive">Inactive</span>
            </div>
            <div class="status-description">Counselor is no longer active
with the service.</div>
            <div class="additional-fields" id="inactiveFields">
              <div class="form-group">
                <label for="inactiveDate">Effective Date</label>
                <input type="date" id="inactiveDate" name="inactiveDate">
              </div>
              <div class="form-group">
                <label for="inactiveReason">Reason</label>
                <select id="inactiveReason" name="inactiveReason">
                  <option value="Resigned">Resigned</option>
                  <option value="Terminated">Terminated</option>
                  <option value="Completed Term">Completed Term</option>
                  <option value="Transfer">Transfer to Another Team</option>
                  <option value="Other">Other</option>
                </select>
              </div>
            </div>
          </div>

          <div class="form-group">
            <label for="statusNotes">Additional Notes</label>
            <textarea id="statusNotes" name="statusNotes" placeholder="Enter
any additional context or notes about this status change..."></textarea>
          </div>

          <div class="form-actions">
```

```
            <button type="button" class="btn btn-secondary"
onclick="cancelUpdate()">Cancel</button>
            <button type="submit" class="btn btn-primary">Update
Status</button>
          </div>
        </form>
      </div>
    </div>
  </div>
  <script>
    // Global variables
    let counselors = [];
    let selectedCounselor = null;

    // Initialize form on load
    document.addEventListener('DOMContentLoaded', function() {
      // Load counselor data
      loadCounselors();

      // Set up form submission
      document.getElementById('updateStatusForm').addEventListener('submit',
submitForm);

      // Set today's date as default for date fields
      setDefaultDates();
    });

    // Load counselors from Google Apps Script
    function loadCounselors() {
      google.script.run
        .withSuccessHandler(populateCounselorDropdown)
        .withFailureHandler(handleError)
        .getCounselorList();
```

```
}

// Populate counselor dropdown
function populateCounselorDropdown(data) {
  counselors = data || [];
  const select = document.getElementById('counselorSelect');

  // Clear existing options (except the first one)
  while (select.options.length > 1) {
    select.remove(1);
  }

  // Add counselor options
  counselors.forEach(counselor => {
    const option = document.createElement('option');
    option.value = counselor.id;
    option.textContent = counselor.name;
    select.appendChild(option);
  });

  // Check if a counselor ID was passed in the URL
  const urlParams = new URLSearchParams(window.location.search);
  const counselorId = urlParams.get('id');
  if (counselorId) {
    select.value = counselorId;
    loadCounselorDetails();
  }
}

// Load counselor details when selected
function loadCounselorDetails() {
  const counselorId = document.getElementById('counselorSelect').value;
  if (!counselorId) {
```

```
        document.getElementById('counselorCard').style.display = 'none';
        document.getElementById('statusForm').style.display = 'none';
        return;
      }

      // Find selected counselor
      selectedCounselor = counselors.find(c => c.id === counselorId);
      if (!selectedCounselor) return;

      // Update counselor card
      document.getElementById('counselorName').textContent =
selectedCounselor.name;
      document.getElementById('counselorTeam').textContent =
selectedCounselor.team || 'N/A';
      document.getElementById('counselorEmail').textContent =
selectedCounselor.email || 'N/A';
      document.getElementById('counselorPhone').textContent =
selectedCounselor.phone || 'N/A';
      document.getElementById('counselorStartDate').textContent =
formatDate(selectedCounselor.startDate) || 'N/A';

      // Set initials for avatar
      const nameParts = selectedCounselor.name.split(' ');
      let initials = '';
      if (nameParts.length >= 2) {
        initials = nameParts[0].charAt(0) + nameParts[1].charAt(0);
      } else {
        initials = nameParts[0].charAt(0);
      }
      document.getElementById('counselorInitials').textContent =
initials.toUpperCase();

      // Update status badge
```

```
    const currentStatus = document.getElementById('currentStatus');
    currentStatus.textContent = selectedCounselor.status || 'Unknown';

    // Update status badge class
    currentStatus.className = 'status-badge';
    switch (selectedCounselor.status) {
      case 'Active': currentStatus.classList.add('status-active'); break;
      case 'Training': currentStatus.classList.add('status-training'); break;
      case 'PTO': currentStatus.classList.add('status-pto'); break;
      case 'UTO': currentStatus.classList.add('status-loa'); break;
      case 'LOA': currentStatus.classList.add('status-loa'); break;
      case 'Inactive': currentStatus.classList.add('status-inactive'); break;
      case 'Overtime': currentStatus.classList.add('status-overtime'); break;
      case 'Disappeared': currentStatus.classList.add('status-disappeared');
break;
      case 'LeavingEarly':
currentStatus.classList.add('status-leaving-early'); break;
      default: currentStatus.classList.add('status-inactive');
    }

    // Show counselor card and status form
    document.getElementById('counselorCard').style.display = 'block';
    document.getElementById('statusForm').style.display = 'block';

    // Reset status selection
    clearStatusSelection();
  }

  // Select a status option
  function selectStatus(element, status) {
    // Clear previous selection
    const options = document.querySelectorAll('.status-option');
    options.forEach(opt => {
```

```
      opt.classList.remove('selected');
      opt.querySelector('input[type="radio"]').checked = false;

      // Hide all additional fields
      const additionalFields = opt.querySelector('.additional-fields');
      if (additionalFields) {
        additionalFields.style.display = 'none';
      }
    });

    // Select current option
    element.classList.add('selected');
    element.querySelector('input[type="radio"]').checked = true;

    // Show additional fields if applicable
    const additionalFields = element.querySelector('.additional-fields');
    if (additionalFields) {
      additionalFields.style.display = 'block';
    }
  }

  // Clear status selection
  function clearStatusSelection() {
    const options = document.querySelectorAll('.status-option');
    options.forEach(opt => {
      opt.classList.remove('selected');
      opt.querySelector('input[type="radio"]').checked = false;

      // Hide all additional fields
      const additionalFields = opt.querySelector('.additional-fields');
      if (additionalFields) {
        additionalFields.style.display = 'none';
      }
```

```
    });
  }

  // Set default dates to today
  function setDefaultDates() {
    const today = new Date().toISOString().split('T')[0];
    const dateInputs = document.querySelectorAll('input[type="date"]');
    dateInputs.forEach(input => {
      input.value = today;
    });
  }

  // Form submission
  function submitForm(e) {
    e.preventDefault();

    // Validate form
    if (!validateForm()) {
      return;
    }

    // Get selected status
    const selectedRadio =
document.querySelector('input[name="status"]:checked');
    if (!selectedRadio) {
      showNotification('Please select a status for the counselor.', 'error');
      return;
    }

    const status = selectedRadio.value;

    // Collect form data based on selected status
    const formData = {
```

```
      counselorId: selectedCounselor.id,
      status: status,
      notes: document.getElementById('statusNotes').value
    };

    // Add status-specific fields
    switch (status) {
      case 'PTO':
        formData.startDate = document.getElementById('ptoStartDate').value;
        formData.endDate = document.getElementById('ptoEndDate').value;
        break;
      case 'UTO':
        formData.date = document.getElementById('utoDate').value;
        formData.shift = document.getElementById('utoShift').value;
        formData.reason = document.getElementById('utoReason').value;
        break;
      case 'Overtime':
        formData.date = document.getElementById('overtimeDate').value;
        formData.hours = document.getElementById('overtimeHours').value;
        formData.reason = document.getElementById('overtimeReason').value;
        break;
      case 'Disappeared':
        formData.lastSeenTime =
document.getElementById('disappearedTime').value;
        formData.actionsTaken =
document.getElementById('disappearedActions').value;
        break;
      case 'LeavingEarly':
        formData.departureTime =
document.getElementById('leavingEarlyTime').value;
        formData.scheduledEndTime =
document.getElementById('scheduledEndTime').value;
```

```
          formData.reason =
document.getElementById('leavingEarlyReason').value;
          break;
        case 'LOA':
          formData.startDate = document.getElementById('loaStartDate').value;
          formData.endDate = document.getElementById('loaEndDate').value;
          formData.type = document.getElementById('loaType').value;
          break;
        case 'Inactive':
          formData.effectiveDate =
document.getElementById('inactiveDate').value;
          formData.reason = document.getElementById('inactiveReason').value;
          break;
      }

      // Submit to Google Apps Script
      google.script.run
        .withSuccessHandler(onSuccess)
        .withFailureHandler(onFailure)
        .updateCounselorStatus(formData);
    }

    // Validate form based on selected status
    function validateForm() {
      const selectedRadio =
document.querySelector('input[name="status"]:checked');
      if (!selectedRadio) {
        showNotification('Please select a status for the counselor.', 'error');
        return false;
      }

      const status = selectedRadio.value;
      let isValid = true;
```

```
    // Validate status-specific fields
    switch (status) {
      case 'PTO':
        const ptoStartDate = document.getElementById('ptoStartDate').value;
        const ptoEndDate = document.getElementById('ptoEndDate').value;
        if (!ptoStartDate || !ptoEndDate) {
          showNotification('Please specify start and end dates for PTO.',
'error');
          isValid = false;
        } else if (new Date(ptoEndDate) < new Date(ptoStartDate)) {
          showNotification('End date cannot be before start date.', 'error');
          isValid = false;
        }
        break;
      case 'Overtime':
        const overtimeHours = document.getElementById('overtimeHours').value;
        if (!overtimeHours || overtimeHours <= 0 || overtimeHours > 24) {
          showNotification('Please enter a valid number of overtime hours
(0.5 - 24).', 'error');
          isValid = false;
        }
        break;
      case 'LOA':
        const loaStartDate = document.getElementById('loaStartDate').value;
        const loaEndDate = document.getElementById('loaEndDate').value;
        if (!loaStartDate) {
          showNotification('Please specify a start date for the leave of
absence.', 'error');
          isValid = false;
        }
        break;
    }
```

```
    return isValid;
  }

  // Success handler
  function onSuccess(response) {
    showNotification('Counselor status updated successfully!', 'success');

    // Optionally, update the counselor's status in the UI
    if (selectedCounselor) {
      const selectedStatus =
document.querySelector('input[name="status"]:checked').value;
      document.getElementById('currentStatus').textContent = selectedStatus;

      // Update status badge class
      const currentStatus = document.getElementById('currentStatus');
      currentStatus.className = 'status-badge';
      switch (selectedStatus) {
        case 'Active': currentStatus.classList.add('status-active'); break;
        case 'PTO': currentStatus.classList.add('status-pto'); break;
        case 'UTO': currentStatus.classList.add('status-loa'); break;
        case 'LOA': currentStatus.classList.add('status-loa'); break;
        case 'Inactive': currentStatus.classList.add('status-inactive');
break;
        case 'Overtime': currentStatus.classList.add('status-overtime');
break;
        case 'Disappeared':
currentStatus.classList.add('status-disappeared'); break;
        case 'LeavingEarly':
currentStatus.classList.add('status-leaving-early'); break;
        default: currentStatus.classList.add('status-inactive');
      }
```

```
      // Clear form
      clearStatusSelection();
      document.getElementById('statusNotes').value = '';
    }

    // Scroll to notification
    document.getElementById('notification').scrollIntoView({ behavior:
'smooth' });
  }

  // Failure handler
  function onFailure(error) {
    showNotification('Error updating status: ' + error.message, 'error');

    // Scroll to notification
    document.getElementById('notification').scrollIntoView({ behavior:
'smooth' });
  }

  // Show notification
  function showNotification(message, type) {
    const notification = document.getElementById('notification');
    notification.textContent = message;
    notification.className = 'notification ' + type;
    notification.style.display = 'block';

    // Hide after 5 seconds
    setTimeout(function() {
      notification.style.display = 'none';
    }, 5000);
  }

  // Cancel update and return to dashboard
```

```
    function cancelUpdate() {
      window.location.href = '?page=dashboard';
    }

    // Format date for display
    function formatDate(dateString) {
      if (!dateString) return '';

      const date = new Date(dateString);
      if (isNaN(date.getTime())) return dateString; // Return as-is if invalid

      const options = { year: 'numeric', month: 'short', day: 'numeric' };
      return date.toLocaleDateString('en-US', options);
    }

    // Handle API errors
    function handleError(error) {
      console.error('Error:', error);
      showNotification('An error occurred: ' + error.message, 'error');
    }
  </script>
</body>
</html>
```