Crisis Support System Documentation for 988 Lifeline LGBTQIA+ Services

System Overview

This documentation provides a comprehensive explanation of the Crisis Support System developed for the 988 Lifeline Crisis Services LGBTQIA+ team. The system is built as a Google Sheets application with Apps Script integration to facilitate crisis support management, team leadership, and operational reporting.

Core Functionality

The Crisis Support System provides comprehensive tools for crisis support operations management including:

1. **Team Management & Counselor Tracking**
    - Adding and monitoring counselor information
    - Status updates and coaching notes
    - 1:1 session documentation
    - Performance tracking
2. **Call Metrics Collection & Reporting**
    - Daily metrics entry and visualization
    - Performance analysis and trends
    - Counselor-specific metrics
    - Report generation for management
3. **Alert Management**
    - Creating and managing alerts by severity
    - Email notifications for critical alerts
    - Resolution tracking
    - Historical alert reporting
4. **Task Management**
    - Creating and assigning tasks
    - Priority-based organization
    - Completion tracking
    - Asana integration for external task management

5. **Team Lead Tools**
    - Shift initialization and tracking
    - Time tracking across different activities
    - Activity breakdowns and reports
    - Performance dashboards
6. **Schedule Management**
    - Weekly schedule initialization
    - Shift assignments and tracking
    - Counselor availability management
    - Schedule reporting

System Structure

The system is organized around several key sheets:

- **Counselor Tracking**: Stores counselor information and status
- **Call Metrics**: Records daily call center performance data
- **Alerts**: Tracks system alerts by severity and status
- **Tasks**: Manages tasks, assignments, and completion status
- **Schedule**: Organizes weekly counselor schedules
- **Team Lead Shifts**: Documents team lead shift information
- **Time Logs**: Records time spent on various activities
- **Coaching Notes**: Stores coaching feedback and follow-ups
- **One-on-One Notes**: Documents 1:1 meetings with counselors
- **Activity Breakdown**: Provides detailed time allocation analysis
- **Time Summary**: Summarizes time usage patterns by week

Team Lead Specific Functions

Shift Management & Coaching

- **Shift Initialization**: Start each shift with structured goals and priorities
- **Team Leadership**: Manage a team of up to 12 Crisis Workers with accountability for performance
- **1:1 Coaching Framework**: Conduct biweekly 30-minute sessions with structured agendas
    - Human check-in (personal wellness)
    - Highlights and wins (reinforcing feedback)
    - Blockers/challenges (identifying growth areas)
    - Biweekly commitments (SMART goals)
    - Counselor reflection

- **Performance Reviews**: Conduct 90-day, mid-year, and annual performance evaluations

Time Management & Tracking

- **Activity Time Tracking**: Document time spent across different activity categories
    - Primary Shift time (direct service)
    - Administrative time
    - Meeting time
    - 1:1 coaching time
    - Other support activities
- **Activity Reflection**: Document successes and challenges with different approaches
- **Productivity Analysis**: View trends in time allocation to optimize team lead effectiveness
- **Counselor Time Management**: Monitor on-queue percentages, after-call work, and call metrics

Service Supervision & Quality

- **On-Shift Supervision**: Provide real-time support to counselors across all teams
- **Metrics Monitoring**: Focus on answer rates (95% target) and on-queue time
- **Quality Evaluation**: Regular review of counselor performance against 70+ quality score target
- **Service Analytics**: Track talk time, handle time, and counselor productivity metrics

Administrative Responsibilities

- **Timecard Management**: Weekly/biweekly timecard approval for team members
- **Attendance Tracking**: Monitor planned and unplanned time off, maintaining below 5% UTO target
- **Documentation Management**: Maintain coaching records in CultureAmp/Asana
- **Weekly Team Meetings**: Lead 45-minute weekly meetings (20min program updates, 25min team activities)

Task & Alert Management

- **Priority Task Handling**: Manage critical issues requiring immediate intervention
- **Task Assignment & Delegation**: Create and assign tasks with clear ownership
- **Alert Management**: Create, monitor, and resolve alerts based on severity level
- **Project Support**: Manage special projects delegated by managers

Data Management

The system maintains several interconnected data structures aligned with the 988 Team Lead role requirements:

1. **Counselor Performance Data**
    - Personal information and status
    - Quality evaluation scores (targeting 70+ average)
    - Answer rates (targeting 95% goal)
    - Call volumes (5-7 conversations per shift for Digital, 8-10 for Lifeline)
    - Talk times and handle times
    - On-queue percentages
    - Time utilization breakdowns
    - Unplanned time off tracking (targeting below 5%)
2. **Coaching & Development Data**
    - Structured 1:1 meeting documentation
    - SMART goal tracking with biweekly commitments
    - Performance review documentation (90-day, mid-year, annual)
    - Feedback history with resolution tracking
    - Development plans with specific milestones
    - Coaching notes with behavior observations
    - Policy & protocol adherence tracking
3. **Team Lead Activity Data**
    - Shift initialization records with established goals
    - Time utilization across activity categories
        - Primary Shift time
        - Administrative time
        - Meeting time
        - 1:1 coaching time
        - Other support activities
    - Effectiveness metrics for different approaches
    - Weekly time summaries with pattern analysis
    - Shift summary reports with accomplishments
4. **Service Oversight Data**
    - Alert management by severity level
    - Task tracking and completion rates
    - Weekly team meeting notes and action items
    - Quality evaluation results by counselor
    - Service-wide metrics for team performance
    - Call pattern analytics for resource planning
    - Policy adherence monitoring

Getting Started for Team Leads

1. **Initial Setup and Onboarding**
    - Run the `initializeSystem()` function to create required sheets
    - Configure email notifications for alerts and critical issues
    - Add your team of counselors (up to 12) to Counselor Tracking
    - Set up individual coaching sections in Asana for each counselor
    - Configure timecard approval settings in ADP
2. **Daily Shift Management**
    - Initialize your shift using "Team Lead Tools > Initialize Shift"
        - Document specific goals for the shift
        - Set service priorities based on current needs
    - Use Time Tracker to categorize and document activities
        - Track Primary Shift, Admin, Meeting, 1:1, and Other time
        - Document efficiency observations for continuous improvement
    - Monitor real-time service metrics (answer rates, on-queue time)
    - Provide on-shift supervision across all counselor teams
    - Create and manage alerts by severity level
    - Update the Attendance Tracker for any counselor absences
3. **Weekly Leadership Activities**
    - Conduct biweekly 30-minute 1:1s with each counselor following the structured format
        - Document all coaching in CultureAmp (or Asana for contractors)
        - Track SMART goal progress and establish new commitments
    - Lead 45-minute weekly team meetings
        - 20 minutes for program updates using Weekly Connection Slide Deck
        - 25 minutes for team building and engagement activities
    - Approve timecards for all team members
        - Full-time staff biweekly through ADP
        - Contractors weekly through timesheet spreadsheet
    - Review quality evaluation results for all counselors
    - Update counselor performance metrics and identify coaching opportunities
4. **Monthly Performance Management**
    - Analyze team metrics against targets
        - Answer rates (95% target)
        - Quality scores (70+ average)
        - Call volumes (5-7 Digital, 8-10 Lifeline per shift)
        - Unplanned time off (below 5% target)
    - Identify performance trends requiring intervention
    - Conduct priority coaching conversations for persistent issues
    - Document performance concerns requiring follow-up
    - Update performance management documentation

- Generate monthly team performance summary for managers

Benefits for Team Management

1. **Enhanced Leadership Effectiveness**
    - Structured framework for coaching up to 12 counselors
    - Centralized view of team performance against clear metrics
    - Early identification of performance issues with quality alerts
    - Time utilization analytics to optimize team lead activities
    - Comprehensive documentation for performance management
2. **Improved Counselor Development**
    - Structured biweekly 1:1 framework with consistent documentation
    - SMART goal tracking with measurable commitments
    - Performance visualization with trend analysis
    - Quality evaluation tracking against 70+ score target
    - Progressive coaching documentation for accountability
3. **Efficient Time Management**
    - Activity categorization across Primary Shift, Admin, Meeting, and 1:1 time
    - Time utilization patterns with effectiveness measurement
    - Automated time tracking with categorized activities
    - Streamlined timecard approval for different employee types
    - Activity reflection tools to identify time optimization opportunities
4. **Comprehensive Service Management**
    - Real-time monitoring of answer rates against 95% target
    - Counselor productivity tracking (5-7 Digital, 8-10 Lifeline conversations)
    - Quality evaluation integration with quality score targets
    - Alert management by severity level with resolution tracking
    - Service pattern analysis to guide resource allocation

Best Practices for Team Leads

1. **Structured Time Management**
    - Initialize shifts with clear goals and expected outcomes
    - Track time across five categories (Primary Shift, Admin, Meeting, 1:1, Other)
    - Document activity effectiveness to continuously refine approaches
    - Balance time between direct supervision and coaching responsibilities
    - Prioritize critical supervision needs while maintaining coaching cadence
    - Document time allocation patterns to identify optimization opportunities

    - Maintain separate tracking for shift goals vs. achievements
2. **Effective Coaching Implementation**

    - Follow the structured 1:1 agenda for all biweekly 30-minute sessions
        - Human check-in → Highlights/wins → Blockers/challenges → Commitments
    - Document all coaching interactions in CultureAmp (Asana for contractors)
    - Focus on behaviors rather than metrics in coaching conversations
    - Create SMART goals with each counselor (limit to 3 active commitments)
    - Document all priority coaching outside of regular 1:1s with email follow-up
    - Approach coaching with curiosity rather than punitive mindset
    - Focus redirection on specific behaviors that are explicitly documented
3. **Quality-Focused Performance Management**

    - Monitor counselor quality scores against 70+ target average
    - Track answer rates with expectation of 95% target
    - Monitor counselor productivity (5-7 Digital, 8-10 Lifeline conversations per shift)
    - Keep unplanned time off below 5% on quarterly basis
    - Review performance regularly against clear, measurable expectations
    - Document all performance concerns with specific examples
    - Maintain progressive coaching documentation with clear expectations
    - Follow up immediately on safety-related concerns
4. **Service Excellence Implementation**

    - Lead weekly 45-minute team meetings with structured format
        - 20 minutes for program updates using Weekly Connection slides
        - 25 minutes for team building and skill development
    - Maintain regular on-shift supervision across all counselor teams
    - Approve timecards weekly (contractors) or biweekly (FT staff)
    - Create alerts for critical issues with appropriate severity assignment
    - Document performance trends that impact service quality
    - Monitor counselor status (on-queue, off-queue, documentation time)
    - Track talk time and handle time trends to identify service improvements

Time Management Subsystem

The Time Management subsystem provides comprehensive tracking and analysis of team lead activities aligned with the 988 Team Lead role requirements:

1. **Team Lead Activity Tracking**

    - **Activity Categorization**: Track time across five core categories
        - Primary Shift time (direct service support)
        - Administrative time (emails, documentation, timecards)

        - Meeting time (team meetings, operations meetings)
        - 1:1 coaching time (biweekly 30-minute sessions)
        - Other support activities (projects, skill camps)
    - **Real-time Activity Logging**: Document activities as they occur
    - **Shift Goal Setting**: Document specific objectives for each shift
    - **Activity Notes**: Record what worked well/poorly for continuous improvement
    - **Activity Breakdown Reports**: View time allocation across categories
    - **Pattern Analysis**: Identify optimization opportunities in time usage
2. **Counselor Time Management**

    - **On-Queue Monitoring**: Track percentage of time counselors are available
    - **Talk Time Analysis**: Monitor average talk time for counselors
    - **After-Call Work**: Track documentation time after conversations
    - **Productivity Metrics**: Monitor conversations per shift against targets
        - Digital staff: 5-7 conversations per shift target
        - Lifeline staff: 8-10 conversations per shift target
    - **Status Analysis**: Break down how counselors allocate their time
    - **Break Adherence**: Monitor break and lunch timing compliance
3. **Coaching Time Optimization**

    - **1:1 Preparation Tracker**: Document time spent preparing for coaching
    - **Coaching Session Timer**: Ensure 30-minute biweekly sessions stay on track
    - **Coaching Note Efficiency**: Track time spent on documentation
    - **Priority Coaching Tracking**: Document time spent on urgent coaching needs
    - **Goal Setting and Follow-up**: Track time spent on SMART goal development
    - **Performance Review Preparation**: Document time spent preparing evaluations
4. **Time Analysis and Reporting**

    - **Weekly Time Summaries**: View aggregated time usage patterns
    - **Activity Effectiveness Rating**: Score different approaches for efficiency
    - **Time vs. Impact Analysis**: Correlate time investments with outcomes
    - **Optimization Recommendations**: Suggest ways to reallocate time
    - **Shift Comparison**: Compare productivity across different shifts and days
    - **Team Lead Benchmarking**: Compare time allocation with other team leads

Management Support Features

The system specifically supports the 988 Team Lead role responsibilities with:

1. **Team Performance Analytics**

    - **Performance Dashboard**: Track key metrics for team of up to 12 counselors

```javascript
// Example counselor productivity analysisconst counselorAnalysis = analyzeCounselorProductivity(counselorData);/* Sample output:Counselor Productivity Analysis==============================Counselor: Alex Johnson (Digital)Performance Metrics:callsPerShift: 5.4 (Target: 6, Adherence: 90.6%)answerRate: 92.3% (Target: 95.0%, Adherence: 97.2%)qualityScore: 68.5 (Target: 70, Adherence: 97.9%)uto: 4.2% (Target: 5.0%, Adherence: 16.0%)Coaching Focus Areas:- All metrics within acceptable ranges*/
```

    - **Target Adherence**: Track four key metrics against targets:
        - Answer Rate (95% target)
        - Quality Evaluation (70+ score target)
        - Productivity (5-7 Digital, 8-10 Lifeline conversations per shift)
        - Unplanned Time Off (below 5% target)
    - **Comparative Analysis**: View performance trends over time
    - **Coaching Focus Identification**: Automatically highlight areas needing attention
2. **Time Management Optimization**

    - **Activity Analysis**: Detailed breakdown of time allocation

```javascript
// Example time allocation analysisconst analysis = analyzeTeamLeadTimeAllocation(sampleTeamLeadData);/* Sample output:Team Lead Time Allocation Analysis==================================Total time tracked: 2400 minutesTime by Category:Primary Shift: 1020 minutes (42.5%)Administrative: 480 minutes (20%)Meetings: 360 minutes (15%)1:1 Coaching: 420 minutes (17.5%)Other: 120 minutes (5%)Recommendations:- Consider increasing time on 1:1 Coaching by approximately 7.5%*/
```

    - **Category Distribution**: Track time across Primary Shift, Admin, Meeting, 1:1, and Other
    - **Optimization Recommendations**: Receive suggestions for better time allocation
    - **Effectiveness Tracking**: Document which approaches yield the best results

3. **Structured Coaching Framework**
    - **1:1 Meeting Template**: Five-section agenda structure
        - Human Check-in
        - Highlights and Wins
        - Blockers/Challenge
        - Biweekly Commitments
        - Counselor Reflection
    - **Performance Documentation**: Integrated with CultureAmp and Asana
    - **Goal Setting Tools**: SMART goal development and tracking
    - **Coaching Email Templates**: Standardized follow-up for priority coaching
4. **Administrative Efficiency**
    - **Timecard Management**: Streamlined approval for both employee types
        - Full-time staff (biweekly in ADP)
        - Contractors (weekly via timesheet spreadsheet)
    - **Attendance Tracking**: Integrated UTO monitoring with alert system
    - **Weekly Team Meeting Tools**: Structured 45-minute format with resources
    - **Report Generation**: Automated reporting for team performance

Support Resources Hub

The Support Resources section serves as a centralized hub for critical documentation and reference materials that team leads need to perform their roles effectively:

1. **Crisis Protocols Documentation**
    - PDF Storage: Repository for all crisis protocols documents
    - Quick Reference: Emergency procedures and immediate actions
    - Categorized Access: Sort protocols by severity and situation type
    - Version Control: Clear indicators of protocol version and last update date
    - Search Functionality: Find specific procedures quickly
2. **Help Resources**
    - System Documentation: Comprehensive guide to using the Crisis Support System
    - Role Documentation: Clear explanation of 988 Team Lead responsibilities
    - FAQ Section: Common questions and troubleshooting advice
    - Video Tutorials: Step-by-step guides for complex processes
    - Contact Information: Support escalation path for system issues
3. **Manager 1:1 Documentation**
    - Meeting Notes: Storage for biweekly manager-team lead 1:1 notes
    - Action Items: Track progress on tasks assigned during manager meetings

    - Resource Sharing: Documents shared by managers for team lead development
    - Goal Tracking: Document and monitor progress on team lead performance goals
    - Feedback Documentation: Store feedback received from managers
4. **Professional Development Resources**

    - Training Materials: Access to training documents and courses
    - Skill Development: Resources for improving coaching and leadership skills
    - Best Practices: Documentation of successful approaches
    - External Resources: Links to valuable external content
    - Continuous Education: Required and recommended learning materials

Comprehensive Menu Structure and Implementation Guide

Overview

This section provides detailed implementation guidance for developers creating the Crisis Support System for 988 Lifeline Crisis Services LGBTQIA+. Each menu section, HTML file requirement, and functional specification is outlined to ensure a seamless implementation that works flawlessly on the first deployment.

Main Menu Structure (`createMainMenu()`)

The main menu should be implemented as follows with specific HTML files required for each function:

1. 📊 Dashboard

- **Function**: `showDashboard()`
- **HTML File**: `dashboard.html`
- **Purpose**: Provide overview of key metrics and system status
- **Content**: Team performance graphs, alert counts, active counselor list, quick links
- **Design**: Clean, card-based layout with metrics prominently displayed
- **Interactive Elements**: Refresh button, drill-down capabilities, date range selector

2. 🗂️ Sidebar

- **Function**: `showSidebar()`
- **HTML File**: `sidebar.html`

- **Purpose**: Quick access to common functions
- **Content**: Compact version of main menu options with key metrics
- **Design**: Vertical layout, collapsible sections, minimal padding
- **Interactive Elements**: Quick action buttons, notifications for pending items

3. 👥 Team Management

This submenu handles all counselor management activities:

a. Add Counselor

- **Function**: `showAddCounselorForm()`
- **HTML File**: `counselor-form.html`
- **Purpose**: Add new counselors to the system
- **Form Fields**: Name, email, phone, status, team, start date, focus/goals, notes
- **Validation**: Required fields, email format, duplicate checking
- **Success Handling**: Clear confirmation, option to add another or return

b. Update Status

- **Function**: `showUpdateStatusForm()`
- **HTML File**: `status-form.html`
- **Purpose**: Update counselor status (active, inactive, leave, etc.)
- **Form Fields**: Counselor selector, status options, effective date, notes
- **Validation**: Required selections, documentation for certain status changes
- **Success Handling**: Clear confirmation with action taken

c. Add Coaching Note

- **Function**: `showCoachingForm()`
- **HTML File**: `coaching-form.html`
- **Purpose**: Document coaching interactions
- **Form Fields**: Counselor selector, coach name, note type, notes, follow-up date
- **Structure**: Match the coaching note structure from documentation
- **Success Handling**: Option to create follow-up task/calendar event

d. Counselor 1:1 Notes

- **Function**: `showOneOnOneNotes()`
- **HTML File**: `one-on-one-notes.html`
- **Purpose**: Document biweekly 1:1 meetings with counselors
- **Structure**: Implement the 5-section format exactly as in documentation:
    1. Human Check-in
    2. Highlights and Wins

3. Blockers/Challenge
4. Biweekly Commitments (SMART goals)
5. Counselor Reflection

- **Design**: Clear section dividers, adequate text space, date tracking
- **Storage**: Save to CultureAmp (full-time staff) or Asana (contractors)

4. 📞 Call Metrics

This submenu handles performance data collection and reporting:

a. Enter Daily Metrics

- **Function**: `showMetricsForm()`
- **HTML File**: `metrics-form.html`
- **Purpose**: Input daily call center performance data
- **Form Fields**: Date, calls offered, calls accepted, talk time, after-call work, on-queue percentage
- **Validation**: Numerical validation, reasonable ranges, date validation
- **Design**: Clear input fields with proper input types (number, date, etc.)

b. View Reports

- **Function**: `showMetricsReport()`
- **HTML File**: `metrics-report.html`
- **Purpose**: View performance reports and trends
- **Content**: Interactive charts, filterable data tables, date range selection
- **Design**: Multiple visualization options, export functionality
- **Interactive Elements**: Drill-down capabilities, comparison tools

5. 🚨 Alerts

This submenu handles alert creation and management:

a. Create Alert

- **Function**: `showAlertForm()`
- **HTML File**: `alert-form.html`
- **Purpose**: Create new system alerts for issues
- **Form Fields**: Severity selector, message, category, status
- **Validation**: Required fields, character limits for messages
- **Design**: Clear severity indicators, character counter

b. View Active Alerts

- **Function**: `showActiveAlerts()`

- **HTML File**: `active-alerts.html`
- **Purpose**: View and manage existing alerts
- **Content**: Filterable list of alerts with status indicators
- **Actions**: Resolve, escalate, reassign functionality
- **Design**: Color-coding by severity, clear status indicators

6. ✅ Tasks

This submenu manages task creation and tracking:

a. Create Task

- **Function**: `showTaskForm()`
- **HTML File**: `task-form.html`
- **Purpose**: Create new tasks for team members
- **Form Fields**: Task name, description, assignee, due date, priority
- **Validation**: Required fields, due date validation
- **Design**: Clear priority indicators, date picker

b. Create Asana Task

- **Function**: `showAsanaTaskForm()`
- **HTML File**: `asana-task-form.html`
- **Purpose**: Create tasks in external Asana system
- **Form Fields**: Task name, description, assignee, due date, priority, Asana project
- **Integration**: Proper API handling with error management
- **Success Handling**: Confirmation with Asana task link

7. 📅 Schedule

This submenu handles schedule management:

a. Manage Schedule

- **Function**: `showScheduleManager()`
- **HTML File**: `schedule-manager.html`
- **Purpose**: Manage team schedules and shifts
- **Content**: Calendar view, shift assignments, time-off indicators
- **Actions**: Assign shifts, approve time-off, adjust schedules
- **Design**: Clear calendar layout, color-coding by shift type

b. Initialize Week

- **Function**: `showInitializeWeekForm()`

- **HTML File**: `initialize-week-form.html`
- **Purpose**: Set up the schedule for a new week
- **Form Fields**: Week start date, shift pattern, template selection
- **Validation**: Date validation, template completeness
- **Actions**: Create shifts based on template or previous week

8. ⏱️ Team Lead Tools

This submenu contains specialized tools for team leads:

a. Initialize Shift

- **Function**: `showShiftInitialization()`
- **HTML File**: `shift-initialization.html`
- **Purpose**: Start a team lead shift with goals and priorities
- **Form Fields**:
  - Shift date & time
  - Shift type
  - Goals for the shift (3-5 fields)
  - Quick notes/priorities
- **Design**: Clean form with adequate text space for goals
- **Success Handling**: Confirmation and option to start time tracker

b. Time Tracker

- **Function**: `showTimeTracker()`
- **HTML File**: `time-tracker.html`
- **Purpose**: Track time spent on different activities
- **Content**:
  - Activity category selection (Primary Shift, Admin, Meeting, 1:1, Other)
  - Timer functionality
  - Activity notes field
  - Previous activities log
- **Design**: Large, easy-to-click buttons for quick category switching
- **Interactive Elements**: Start/stop timers, quick category switching
- **Analysis**: Show time distribution with recommendations

9. 🆘 Support Resources

This submenu provides access to documentation and resources:

a. Crisis Protocols

- **Function**: `showCrisisProtocols()`

- **HTML File**: `crisis-protocols.html`
- **Purpose**: Central repository for crisis response protocols
- **Content**:
    - Categorized list of PDF protocols
    - Search functionality
    - Quick reference guides
    - Version information and update dates
- **Design**: Clean document library with clear categorization
- **File Handling**: PDF viewer integration, download options

b. Help Resources

- **Function**: `showHelp()`
- **HTML File**: `help.html`
- **Purpose**: System help and documentation
- **Content**:
    - User guide
    - FAQ section
    - Video tutorials
    - Troubleshooting guide
    - Contact information
- **Design**: Searchable, categorized help content
- **Interactive Elements**: Step-by-step guides, expandable sections

c. Manager 1:1 Documentation (New)

- **Function**: `showManagerOneOnOnes()`
- **HTML File**: `manager-one-on-ones.html`
- **Purpose**: Store and manage team lead's 1:1s with their manager
- **Content**:
    - Meeting date and notes
    - Action items and follow-ups
    - Performance feedback
    - Goal tracking
    - Resource links shared by manager
- **Design**: Similar to counselor 1:1 notes but focused on team lead development
- **Privacy**: Access restricted to the team lead and their manager

10. ⚙️ Settings

- **Function**: `showSettings()`
- **HTML File**: `settings.html`
- **Purpose**: Configure system settings
- **Content**: Email notifications, Asana integration, display preferences

- **Design**: Clear sections with appropriate input types
- **Validation**: API key format checking, email validation
- **Success Handling**: Save confirmation, testing options for integrations

Trevor Project Inspired Design System

Based on the beautiful illustrations and brand colors you've shared, we should incorporate The Trevor Project's visual identity into the Crisis Support System while modernizing it for digital use. Below is a comprehensive design system that maintains brand consistency while creating a calming, supportive interface for team leads.

Color Palette

Primary Colors

- **Trevor Orange**: #FF786E (HEX: #FF786E, CMYK: 0,67,51,0, RGB: 255,120,110)
- **Deep Blue**: #001A4E (HEX: #001A4E, CMYK: 100,93,34,43, RGB: 0,26,78)
- **Purple**: #9A3499 (HEX: #9A3499, CMYK: 39,93,0,0, RGB: 155,52,153)
- **Teal**: #137F6A (HEX: #137F6A, CMYK: 86,29,66,11, RGB: 19,127,106)

Secondary Colors

- **Light Blue**: #4F52DE (HEX: #4F52DE, CMYK: 69,0,11,0, RGB: 79,197,222)
- **Soft Yellow**: #FFAD8D (HEX: #FFAD8D, CMYK: 0,45,40,0, RGB: 255,168,141)
- **Lavender**: #B3AE4A (HEX: #B3AE4A, CMYK: 31,27,93,0, RGB: 179,174,74)
- **Light Purple**: #D58AC (HEX: #F54AC, CMYK: 0,97,3,0, RGB: 245,11,139)

Tertiary/Accent Colors

- **Soft Green**: #BAE2CE (HEX: #BAE2CE, CMYK: 27,0,19,0, RGB: 186,226,206)
- **Pale Yellow**: #FFF2DF (HEX: #FFF2DF, CMYK: 0,3,9,0, RGB: 255,242,223)
- **Light Pink**: #FBCBBE (HEX: #FBCBBE, CMYK: 0,23,20,0, RGB: 251,203,190)
- **Soft Lavender**: #D1CFCC (HEX: #D1CFCC, CMYK: 17,11,14,0, RGB: 209,207,204)

Functional Colors

- **Success**: #137F6A (Teal)
- **Warning**: #FFAD8D (Soft Yellow)
- **Error/Alert**: #F54AC (Light Purple)
- **Info**: #4F52DE (Light Blue)

Gradient Combinations

To create the soothing, optimistic effect mentioned:

1. **Primary Gradient**:
    - From #FF786E (Trevor Orange) to #FBCBBE (Light Pink)
    - Direction: 135deg (top-left to bottom-right)
2. **Calm Gradient**:
    - From #4F52DE (Light Blue) to #BAE2CE (Soft Green)
    - Direction: 135deg (top-left to bottom-right)
3. **Support Gradient**:
    - From #9A3499 (Purple) to #D58AC (Light Purple)
    - Direction: 45deg (bottom-left to top-right)
4. **Background Gradient**:
    - From #FFF2DF (Pale Yellow) to #FFFFFF (White)
    - Direction: 180deg (top to bottom)

Typography

- **Primary Font**: Roboto
- **Header Font**: Poppins (for a modern, friendly feel)
- **Font Weights**:
    - Headers: 600 (semibold)
    - Subheaders: 500 (medium)
    - Body: 400 (regular)
    - Emphasis: 700 (bold)

Illustration Integration

The illustrations you've shared showcase diverse individuals in supportive scenarios, using bold colors and simplified forms. These should be integrated throughout the system:

1. **Dashboard**: Use Image 3 (people communicating) as a background element or welcome graphic
2. **Support Resources**: Use Image 1 (person self-soothing) to emphasize self-care
3. **Team Management**: Use Image 2 (person with phone) to represent connectivity
4. **Login/Welcome**: Use all illustrations in a carousel to represent diversity and support

UI Component Styling

Cards

- Soft rounded corners (12px radius)
- Light shadow: 0 4px 8px rgba(0, 0, 0, 0.05)
- White background or very pale gradient
- Orange accent line on the left or top (3px)

Buttons

- Primary: Trevor Orange gradient with white text
- Secondary: White with Trevor Orange border and text
- Danger: Light Purple with

headers: ['Date', 'Shift Type', 'Start Time', 'End Time', 'Goals', 'Quick Notes', 'Created By', 'Timestamp'] });

```
Unset
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') .addItem('Run Error Diagnostics', 'runErrorDiagnostics')); }

/**

- HTML for the System Validation Section in Dashboard */ function createValidationStatusHtml() { return `
    <h3>System Health</h3> <button class="btn-primary btn-sm" onclick="runValidation()">Validate Now</button>   System Health --   Last Validation --   0 Critical   0 Warnings   <script> // Load validation status when dashboard loads function loadValidationStatus() { google.script.run .withSuccessHandler(updateValidationStatus) .withFailureHandler(handleError) .getValidationStatus(); } // Run validation from the dashboard function runValidation() { // Show loading state document.getElementById('validationStatus').innerHTML = ' 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 = ' Healthy'; } else if (status.criticalIssues > 0) { statusElement.innerHTML = ' Critical Issues'; } else { statusElement.innerHTML = ' 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'; } } </script>

`; }

Interactive HTML Components Implementation Guide

The interactive HTML components of the Crisis Support System should be implemented using modern CSS and JavaScript techniques to provide a seamless user experience. Below are detailed implementations for key interactive elements that should be included in the system.

1. Interactive Time Tracker

The Time Tracker is one of the most critical interface components for team leads. Here's a detailed implementation guide with CSS, HTML, and JavaScript:

Unset

```
<!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;40
0;500;700&family=Poppins:wght@500;600;700&display=swap"
rel="stylesheet">
  <style>
    /* Variables for Trevor Project color scheme */
    :root {
      --trevor-orange: #FF786E;
      --deep-blue: #001A4E;
      --purple: #9A3499;
      --teal: #137F6A;
      --light-blue: #4F52DE;
      --soft-yellow: #FFAD8D;
      --soft-green: #BAE2CE;
      --light-pink: #FBCBBE;
      --pale-yellow: #FFF2DF;

      --primary-gradient: linear-gradient(135deg,
var(--trevor-orange), var(--light-pink));
      --calm-gradient: linear-gradient(135deg, var(--light-blue),
var(--soft-green));
```

```
  --spacing-xs: 4px;
  --spacing-sm: 8px;
  --spacing-md: 16px;
  --spacing-lg: 24px;
  --spacing-xl: 32px;

  --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: 1000px;
  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);
  padding-bottom: var(--spacing-md);
  border-bottom: 1px solid rgba(0, 0, 0, 0.1);
}
```

```css
/* Timer display styles */
.timer-container {
  display: flex;
  flex-direction: column;
  align-items: center;
  margin-bottom: var(--spacing-xl);
}

.timer-display {
  background: white;
  border-radius: var(--border-radius-lg);
  padding: var(--spacing-lg);
  box-shadow: 0 4px 16px rgba(0, 0, 0, 0.1);
  text-align: center;
  margin-bottom: var(--spacing-lg);
  width: 100%;
  max-width: 400px;
}

.time {
  font-size: 48px;
  font-weight: 700;
  color: var(--deep-blue);
  font-family: 'Poppins', sans-serif;
}

.activity-label {
  font-size: 18px;
  color: var(--trevor-orange);
  margin-top: var(--spacing-md);
}

.timer-controls {
  display: flex;
  gap: var(--spacing-md);
```

```
      margin-top: var(--spacing-lg);
    }

    /* Category selection styles */
    .category-buttons {
      display: grid;
      grid-template-columns: repeat(auto-fit, minmax(150px,
1fr));
      gap: var(--spacing-md);
      margin-bottom: var(--spacing-xl);
    }

    .category-button {
      display: flex;
      flex-direction: column;
      align-items: center;
      justify-content: center;
      height: 120px;
      background: white;
      border-radius: var(--border-radius-md);
      padding: var(--spacing-md);
      text-align: center;
      border: 2px solid transparent;
      cursor: pointer;
      transition: all 0.3s ease;
    }

    .category-button.active {
      border-color: var(--trevor-orange);
      box-shadow: 0 4px 12px rgba(255, 120, 110, 0.2);
    }

    .category-button:hover {
      transform: translateY(-2px);
      box-shadow: 0 6px 12px rgba(0, 0, 0, 0.1);
    }
```

```
.category-icon {
  width: 40px;
  height: 40px;
  display: flex;
  align-items: center;
  justify-content: center;
  border-radius: 50%;
  margin-bottom: var(--spacing-sm);
}

.primary-shift .category-icon {
  background-color: rgba(255, 120, 110, 0.2);
  color: var(--trevor-orange);
}

.administrative .category-icon {
  background-color: rgba(155, 52, 153, 0.2);
  color: var(--purple);
}

.meetings .category-icon {
  background-color: rgba(79, 82, 222, 0.2);
  color: var(--light-blue);
}

.coaching .category-icon {
  background-color: rgba(19, 127, 106, 0.2);
  color: var(--teal);
}

.other .category-icon {
  background-color: rgba(255, 168, 141, 0.2);
  color: var(--soft-yellow);
}
```

```
    .category-name {
      font-weight: 500;
      margin-bottom: var(--spacing-xs);
    }

    .category-description {
      font-size: 12px;
      color: #666;
    }

    /* Notes section */
    .notes-section {
      margin-bottom: var(--spacing-xl);
    }

    textarea {
      width: 100%;
      min-height: 100px;
      border: 1px solid #ddd;
      border-radius: var(--border-radius-sm);
      padding: var(--spacing-md);
      font-family: 'Roboto', sans-serif;
      resize: vertical;
    }

    /* Summary and recent activities */
    .summary-section {
      margin-bottom: var(--spacing-xl);
    }

    .summary-grid {
      display: grid;
      grid-template-columns: repeat(auto-fit, minmax(150px,
1fr));
      gap: var(--spacing-md);
      margin-bottom: var(--spacing-lg);
```

```
}

.summary-item {
  background: white;
  border-radius: var(--border-radius-sm);
  padding: var(--spacing-md);
  text-align: center;
}

.summary-value {
  font-size: 24px;
  font-weight: 700;
  margin-bottom: var(--spacing-xs);
}

.summary-label {
  font-size: 14px;
  color: #666;
}

.activities-section {
  margin-bottom: var(--spacing-xl);
}

.activity-list {
  background: white;
  border-radius: var(--border-radius-md);
  box-shadow: 0 2px 8px rgba(0, 0, 0, 0.05);
  overflow: hidden;
}

.activity-item {
  padding: var(--spacing-md);
  border-bottom: 1px solid #eee;
}
```

```
.activity-item:last-child {
  border-bottom: none;
}

.activity-header {
  display: flex;
  justify-content: space-between;
  margin-bottom: var(--spacing-xs);
}

.activity-type {
  font-weight: 500;
}

.activity-time {
  color: #666;
  font-size: 14px;
}

.activity-duration {
  font-weight: 700;
  margin-bottom: var(--spacing-xs);
  color: var(--trevor-orange);
}

.activity-notes {
  font-size: 14px;
  color: #444;
}

/* Buttons */
.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-large {
  font-size: 18px;
  padding: var(--spacing-md) var(--spacing-xl);
}

/* 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);
```

```
    }

    /* Animations */
    @keyframes pulse {
      0% { transform: scale(1); }
      50% { transform: scale(1.05); }
      100% { transform: scale(1); }
    }

    .timer-active .time {
      animation: pulse 2s infinite;
      color: var(--trevor-orange);
    }
  </style>
</head>

<body>
  <div class="container">
    <div class="header">
      <h1>Team Lead Time Tracker</h1>
      <div id="currentDate"></div>
    </div>

    <div class="timer-container">
      <div class="timer-display" id="timerDisplay">
        <div class="time" id="timer">00:00:00</div>
        <div class="activity-label" id="activityLabel">Not
tracking</div>
      </div>

      <div class="timer-controls">
        <button id="startBtn" class="btn btn-primary btn-large"
onclick="startTimer()">Start</button>
        <button id="pauseBtn" class="btn btn-secondary btn-large"
onclick="pauseTimer()" disabled>Pause</button>
```

```
        <button id="stopBtn" class="btn btn-secondary btn-large"
onclick="stopTimer()" disabled>Stop</button>
      </div>
    </div>

    <h2>Select Activity Category</h2>
    <div class="category-buttons">
      <div class="category-button primary-shift"
onclick="selectCategory('Primary Shift')">
        <div class="category-icon">👥</div>
        <div class="category-name">Primary Shift</div>
        <div class="category-description">Direct service
support</div>
      </div>

      <div class="category-button administrative"
onclick="selectCategory('Administrative')">
        <div class="category-icon">📋</div>
        <div class="category-name">Administrative</div>
        <div class="category-description">Documentation,
emails</div>
      </div>

      <div class="category-button meetings"
onclick="selectCategory('Meetings')">
        <div class="category-icon">🗓️</div>
        <div class="category-name">Meetings</div>
        <div class="category-description">Team & ops
meetings</div>
      </div>

      <div class="category-button coaching"
onclick="selectCategory('1:1 Coaching')">
        <div class="category-icon">🤝</div>
        <div class="category-name">1:1 Coaching</div>
```

```
        <div class="category-description">Individual
sessions</div>
      </div>

      <div class="category-button other"
onclick="selectCategory('Other')">
        <div class="category-icon">✨</div>
        <div class="category-name">Other</div>
        <div class="category-description">Special projects</div>
      </div>
    </div>

    <div class="notes-section">
      <h2>Activity Notes</h2>
      <textarea id="activityNotes" placeholder="Describe what
you're working on and what's working well/not
working..."></textarea>
    </div>

    <div class="summary-section">
      <h2>Today's Summary</h2>
      <div class="summary-grid">
        <div class="summary-item">
          <div class="summary-value"
id="primaryShiftTime">00:00:00</div>
          <div class="summary-label">Primary Shift</div>
        </div>

        <div class="summary-item">
          <div class="summary-value"
id="adminTime">00:00:00</div>
          <div class="summary-label">Administrative</div>
        </div>

        <div class="summary-item">
```

```
          <div class="summary-value"
id="meetingsTime">00:00:00</div>
          <div class="summary-label">Meetings</div>
        </div>

        <div class="summary-item">
          <div class="summary-value"
id="coachingTime">00:00:00</div>
          <div class="summary-label">1:1 Coaching</div>
        </div>

        <div class="summary-item">
          <div class="summary-value"
id="otherTime">00:00:00</div>
          <div class="summary-label">Other</div>
        </div>

        <div class="summary-item">
          <div class="summary-value"
id="totalTime">00:00:00</div>
          <div class="summary-label">Total</div>
        </div>
      </div>
    </div>

    <div class="activities-section">
      <h2>Recent Activities</h2>
      <div id="activitiesList" class="activity-list">
        <!-- Activities will be populated here -->
        <div class="loading">Loading recent activities...</div>
      </div>
    </div>

    <div class="footer">
      <button class="btn btn-secondary"
onclick="google.script.host.close()">Close</button>
```

```
      <button class="btn btn-primary"
onclick="saveTimeLog()">Save Time Log</button>
    </div>
  </div>

  <script>
    // Timer variables
    let timerInterval;
    let seconds = 0;
    let isRunning = false;
    let currentCategory = '';
    let activityStart = null;
    let todaysActivities = [];
    let categorySummary = {
      'Primary Shift': 0,
      'Administrative': 0,
      'Meetings': 0,
      '1:1 Coaching': 0,
      'Other': 0
    };

    // Initialize on page load
    document.addEventListener('DOMContentLoaded', function() {
      // Set current date
      const today = new Date();
      document.getElementById('currentDate').innerText =
today.toLocaleDateString();

      // Load today's data
      loadTodaysTimeLog();
      loadRecentActivities();
    });

    // Select an activity category
    function selectCategory(category) {
      // If timer is running, stop current activity first
```

```
      if (isRunning) {
        stopTimer();
      }

      // Set new category
      currentCategory = category;
      document.getElementById('activityLabel').innerText =
category;

      // Clear any previous selection
      document.querySelectorAll('.category-button').forEach(btn
=> {
        btn.classList.remove('active');
      });

      // Highlight selected button

document.querySelector(`.category-button.${category.toLowerCase()
.replace(/\s+/g, '-').replace(/[:]/g,
'')}`).classList.add('active');

      // Enable start button
      document.getElementById('startBtn').disabled = false;
    }

    // Start the timer
    function startTimer() {
      if (!currentCategory) {
        alert('Please select an activity category first');
        return;
      }

      // Record start time if not already running
      if (!isRunning) {
        activityStart = new Date();
        isRunning = true;
```

```
      }

      // Update UI

document.getElementById('timerDisplay').classList.add('timer-acti
ve');
      document.getElementById('startBtn').disabled = true;
      document.getElementById('pauseBtn').disabled = false;
      document.getElementById('stopBtn').disabled = false;

      // Start timer
      timerInterval = setInterval(updateTimer, 1000);
    }

    // Update the timer display
    function updateTimer() {
      seconds++;
      const h = Math.floor(seconds / 3600);
      const m = Math.floor((seconds % 3600) / 60);
      const s = seconds % 60;

      document.getElementById('timer').innerText =
        `${h.toString().padStart(2,
'0')}:${m.toString().padStart(2, '0')}:${s.toString().padStart(2,
'0')}`;
    }

    // Pause the timer
    function pauseTimer() {
      clearInterval(timerInterval);
      isRunning = false;

      // Update UI

document.getElementById('timerDisplay').classList.remove('timer-a
ctive');
```

```
      document.getElementById('startBtn').disabled = false;
      document.getElementById('pauseBtn').disabled = true;
    }

    // Stop the timer and save the activity
    function stopTimer() {
      // Only process if we have an activity and time
      if (currentCategory && seconds > 0) {
        const activityEnd = new Date();
        const duration = seconds;
        const notes =
document.getElementById('activityNotes').value;

        // Add to activities list
        todaysActivities.push({
          category: currentCategory,
          startTime: activityStart,
          endTime: activityEnd,
          durationSeconds: duration,
          notes: notes
        });

        // Update category summary
        categorySummary[currentCategory] += duration;
        updateSummary();

        // Save to server
        recordActivity(currentCategory, activityStart,
activityEnd, duration, notes);

        // Add to activity list in UI
        addActivityToList({
          activityType: currentCategory,
          startTime: activityStart.toLocaleTimeString(),
          endTime: activityEnd.toLocaleTimeString(),
          durationSeconds: duration,
```

```
          notes: notes
        });

        // Reset activity notes
        document.getElementById('activityNotes').value = '';
      }

      // Reset timer
      clearInterval(timerInterval);
      seconds = 0;
      isRunning = false;
      document.getElementById('timer').innerText = '00:00:00';
      document.getElementById('activityLabel').innerText = 'Not
tracking';

      // Update UI

document.getElementById('timerDisplay').classList.remove('timer-a
ctive');
      document.getElementById('startBtn').disabled = false;
      document.getElementById('pauseBtn').disabled = true;
      document.getElementById('stopBtn').disabled = true;

      // Deselect category buttons
      document.querySelectorAll('.category-button').forEach(btn
=> {
        btn.classList.remove('active');
      });

      currentCategory = '';
      activityStart = null;
    }

    // Record activity on the server
    function recordActivity(activityType, startTime, endTime,
durationSeconds, notes) {
```

```
      const activityData = {
        date: new Date().toISOString().split('T')[0],
        activityType: activityType,
        startTime: startTime.toLocaleTimeString(),
        endTime: endTime.toLocaleTimeString(),
        durationSeconds: durationSeconds,
        notes: notes
      };

      google.script.run
        .withSuccessHandler(onActivityRecorded)
        .withFailureHandler(handleError)
        .recordActivityTimestamp(activityData);
    }

    // Callback when activity is recorded
    function onActivityRecorded(result) {
      // Activity recorded successfully
    }

    // Update the summary display
    function updateSummary() {
      // Update each category time display
      document.getElementById('primaryShiftTime').innerText =
formatTime(categorySummary['Primary Shift']);
      document.getElementById('adminTime').innerText =
formatTime(categorySummary['Administrative']);
      document.getElementById('meetingsTime').innerText =
formatTime(categorySummary['Meetings']);
      document.getElementById('coachingTime').innerText =
formatTime(categorySummary['1:1 Coaching']);
      document.getElementById('otherTime').innerText =
formatTime(categorySummary['Other']);

      // Update total time
```

```
      const totalSeconds =
Object.values(categorySummary).reduce((sum, time) => sum + time,
0);
      document.getElementById('totalTime').innerText =
formatTime(totalSeconds);
    }

    // Format time from seconds to HH:MM:SS
    function formatTime(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')}`;
    }

    // Load today's time log from the server
    function loadTodaysTimeLog() {
      google.script.run
        .withSuccessHandler(processTodaysLog)
        .withFailureHandler(handleError)
        .getTodaysTimeLog();
    }

    // Process today's log data
    function processTodaysLog(logData) {
      if (logData.found) {
        // Process existing time log data
        categorySummary = {
          'Primary Shift': logData.times['Primary Shift'] || 0,
          'Administrative': logData.times['Administrative'] || 0,
          'Meetings': logData.times['Meetings'] || 0,
          '1:1 Coaching': logData.times['1:1 Coaching'] || 0,
          'Other': logData.times['Other'] || 0
```

```
        };
        updateSummary();
      }
    }

    // Load recent activities from the server
    function loadRecentActivities() {
      google.script.run
        .withSuccessHandler(displayActivities)
        .withFailureHandler(handleError)
        .getRecentActivities();
    }

    // Display recent activities
    function displayActivities(activities) {
      const container =
document.getElementById('activitiesList');
      container.innerHTML = '';

      if (!activities || activities.length === 0) {
        container.innerHTML = '<div class="activity-item">No
recent activities recorded today</div>';
        return;
      }

      activities.forEach(activity => {
        addActivityToList(activity);
      });
    }

    // Add a single activity to the list
    function addActivityToList(activity) {
      const container =
document.getElementById('activitiesList');

      if (container.querySelector('.loading')) {
```

```
        container.innerHTML = '';
      }

      const activityEl = document.createElement('div');
      activityEl.className = 'activity-item';

      activityEl.innerHTML = `
        <div class="activity-header">
          <span
class="activity-type">${activity.activityType}</span>
          <span class="activity-time">${activity.startTime} -
${activity.endTime}</span>
        </div>
        <div
class="activity-duration">${formatTime(activity.durationSeconds)}
</div>
        <div class="activity-notes">${activity.notes || 'No
notes'}</div>
      `;

      container.insertBefore(activityEl, container.firstChild);
    }

    // Save    try {
      const ss = SpreadsheetApp.getActiveSpreadsheet();
      const protection =
ss.getProtections(SpreadsheetApp.ProtectionType.SHEET);

      // Check if critical sheets are protected
      const criticalSheets = [
        CRISIS_SUPPORT_CONFIG.SHEETS.ERROR_LOG,
        CRISIS_SUPPORT_CONFIG.SHEETS.SYSTEM_LOG
      ];

      const protectedSheetNames = protection.map(p =>
p.getSheet().getName());
```

```
      const unprotectedCriticalSheets =
criticalSheets.filter(name =>
        !protectedSheetNames.includes(name) &&
ss.getSheetByName(name)
      );

      // Create check result
      const checkResult = {
        name: 'Critical Sheet Protection',
        passed: unprotectedCriticalSheets.length === 0,
        level: this.LEVELS.WARNING,
        message: unprotectedCriticalSheets.length === 0
          ? 'All critical sheets are protected'
          : `Found ${unprotectedCriticalSheets.length}
unprotected critical sheets`,
        fixable: true,
        fixed: false,
        fixMessage: 'Add protection to critical sheets',
        details: `Unprotected critical sheets:
${unprotectedCriticalSheets.join(', ')}`
      };

      // If there are issues and we should fix them
      if (!checkResult.passed && (autoFix || (interactive &&
this.shouldFixInteractively(checkResult)))) {
        try {
          // Add protection to each unprotected critical sheet
          unprotectedCriticalSheets.forEach(sheetName => {
            const sheet = ss.getSheetByName(sheetName);
            if (sheet) {
              const sheetProtection =
sheet.protect().setDescription('System Protection');

              // Get the current user as an editor
              const me = Session.getEffectiveUser();
              sheetProtection.addEditor(me);
```

```
              // Remove all other editors to make it exclusive to
you

sheetProtection.removeEditors(sheetProtection.getEditors());
              // But add yourself back as an editor
              sheetProtection.addEditor(me);
            }
          });

          // Update check result
          checkResult.fixed = true;
          checkResult.message = `Added protection to
${unprotectedCriticalSheets.length} critical sheets`;

this.validationState.validationResults[this.CATEGORIES.ACCESS_PER
MISSIONS].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.ACCESS_PER
MISSIONS].checks.push(checkResult);

      // Increment issues count if not passed
      if (!checkResult.passed) {

this.validationState.validationResults[this.CATEGORIES.ACCESS_PER
MISSIONS].issues++;
        this.validationState.issuesFound++;
```

```
      }
    } catch (e) {
      // Create error check result
      const errorCheck = {
        name: 'Permissions Check Error',
        passed: false,
        level: this.LEVELS.INFO,
        message: `Error checking permissions: ${e.message}`,
        fixable: false,
        fixed: false,
        details: e.stack
      };

      // Add the check result

this.validationState.validationResults[this.CATEGORIES.ACCESS_PER
MISSIONS].checks.push(errorCheck);

      // Increment issues count

this.validationState.validationResults[this.CATEGORIES.ACCESS_PER
MISSIONS].issues++;
      this.validationState.issuesFound++;
    }
  },

  /**
   * Validates formulas in the spreadsheet
   */
  validateFormulas: function(interactive, autoFix) {
    // Initialize results object for this category

this.validationState.validationResults[this.CATEGORIES.FORMULA_HE
ALTH] = {
      checks: [],
      issues: 0,
```

```
      fixed: 0
    };

    // Update progress indicator if interactive
    if (interactive) {
      this.updateProgressIndicator('Checking formulas...', 90);
    }

    try {
      const ss = SpreadsheetApp.getActiveSpreadsheet();
      const sheets = ss.getSheets();

      let errorCells = [];

      // Check each sheet for formula errors
      sheets.forEach(sheet => {
        const sheetName = sheet.getName();
        const dataRange = sheet.getDataRange();

        // Skip empty sheets
        if (dataRange.getNumRows() <= 1 &&
dataRange.getNumColumns() <= 1) {
          return;
        }

        // Get formula cells
        const formulas = dataRange.getFormulas();
        const values = dataRange.getValues();

        // Check each cell for formula errors
        for (let row = 0; row < formulas.length; row++) {
          for (let col = 0; col < formulas[row].length; col++) {
            // If it's a formula
            if (formulas[row][col] &&
formulas[row][col].startsWith('=')) {
              const value = values[row][col];
```

```
              // Check for common error values
              if (value === '#N/A' || value === '#DIV/0!' ||
value === '#VALUE!' ||
                  value === '#REF!' || value === '#NUM!' || value
=== '#NAME?') {
                errorCells.push({
                  sheet: sheetName,
                  row: row + 1,
                  column: col + 1,
                  formula: formulas[row][col],
                  error: value
                });
              }
            }
          }
        }
      });

      // Create check result
      const checkResult = {
        name: 'Formula Error Check',
        passed: errorCells.length === 0,
        level: this.LEVELS.WARNING,
        message: errorCells.length === 0
          ? 'No formula errors found'
          : `Found ${errorCells.length} cells with formula
errors`,
        fixable: false, // Complex fix requiring analysis
        fixed: false,
        fixMessage: 'Review and fix formula errors',
        details: errorCells.map(cell =>
          `Sheet: ${cell.sheet}, Cell:
R${cell.row}C${cell.column}, Error: ${cell.error}, Formula:
${cell.formula}`
        ).join('\n')
      };
```

```
      // Add the check result

this.validationState.validationResults[this.CATEGORIES.FORMULA_HE
ALTH].checks.push(checkResult);

      // Increment issues count if not passed
      if (!checkResult.passed) {

this.validationState.validationResults[this.CATEGORIES.FORMULA_HE
ALTH].issues++;
        this.validationState.issuesFound++;
      }
    } catch (e) {
      // Create error check result
      const errorCheck = {
        name: 'Formula Check Error',
        passed: false,
        level: this.LEVELS.INFO,
        message: `Error checking formulas: ${e.message}`,
        fixable: false,
        fixed: false,
        details: e.stack
      };

      // Add the check result

this.validationState.validationResults[this.CATEGORIES.FORMULA_HE
ALTH].checks.push(errorCheck);

      // Increment issues count

this.validationState.validationResults[this.CATEGORIES.FORMULA_HE
ALTH].issues++;
      this.validationState.issuesFound++;
    }
```

```
  },

  /**
   * 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 = UIHandler.getUi();
    if (!ui) return false;

    // Create message based on severity
    let icon = '⚠️';
    if (issue.level === this.LEVELS.CRITICAL) icon = '🔴';
```

```
    else if (issue.level === this.LEVELS.INFO) icon = 'ℹ️';

    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();
                }
              })
              .withFailureHandler(function(error) {
                document.getElementById('message').innerText =
'Error: ' + error.message;
              })
              .listenForProgressUpdates();
          </script>
        </body>
      </html>
    `)
    .setWidth(400)
    .setHeight(200);

    try {
      UIHandler.showModelessDialog(html, 'System Validation');
    } catch (e) {
      console.error('Could not show progress indicator:', e);
    }
```

```
  },

  /**
   * Updates the progress indicator
   * @param {string} message - The message to display
   * @param {number} percent - The progress percentage
   */
  updateProgressIndicator: function(message, percent) {
    // This would typically update a global variable that's
checked by a
    // function called by the progress dialog

PropertiesService.getScriptProperties().setProperty('validationPr
ogress',
      JSON.stringify({
        message: message,
        percent: percent
      })
    );
  },

  /**
   * Hides the progress indicator
   */
  hideProgressIndicator: function() {
    // Set a flag to close the dialog

PropertiesService.getScriptProperties().setProperty('validationPr
ogress',
      JSON.stringify({
        close: true
      })
    );
  },

  /**
```

```
   * Function called by the progress dialog to get updates
   */
  listenForProgressUpdates: function() {
    const progressData =
PropertiesService.getScriptProperties().getProperty('validationPr
ogress');
    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);

    try {
      UIHandler.showModalDialog(html, 'System Validation
Results');
    } catch (e) {
      console.error('Could not show validation results:', e);
    }
```

```
  },

  /**
   * Shows an error that occurred during validation
   * @param {Error} error - The error that occurred
   */
  showValidationError: function(error) {
    const ui = UIHandler.getUi();
    if (!ui) {
      console.error('Validation error:', error);
      return;
    }

    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',
'### Buttons
- Primary: Trevor Orange gradient with white text
- Secondary: White with Trevor Orange border and text
```

- Danger: Light Purple with white text
- Rounded corners (8px radius)
- Hover effect: Slight darkening (15%) with 0.2s transition

Forms
- Input fields with rounded corners (8px)
- Light gray borders (#E0E0E0)
- Orange focus state
- Clear, left-aligned labels
- Inline validation with appropriate colors

Navigation
- Left sidebar with Deep Blue background
- White text for readability
- Orange indicator for active section
- Icons with 24px size for clear visibility

Animation & Transitions

- Subtle fade-in for page loads (0.3s)
- Smooth transitions between states (0.2s)
- Gentle hover effects on interactive elements
- Loading states using gradient animation

Implementation in HTML

When implementing this design system in the HTML files:

```html
<style>
  :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);

    /* Typography */
    --font-primary: 'Roboto', sans-serif;
    --font-header: 'Poppins', sans-serif;

    /* 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: var(--font-primary);
  background: var(--background-gradient);
  color: var(--deep-blue);
  margin: 0;
  padding: 0;
}

h1, h2, h3, h4, h5, h6 {
  font-family: var(--font-header);
  color: var(--deep-blue);
}

.btn-primary {
  background: var(--primary-gradient);
  color: white;
  border: none;
  border-radius: var(--border-radius-sm);
  padding: var(--spacing-sm) var(--spacing-md);
  transition: all 0.2s ease;
}

.btn-primary:hover {
  box-shadow: 0 4px 8px rgba(255, 120, 110, 0.3);
  transform: translateY(-1px);
}

.card {
  background: white;
  border-radius: var(--border-radius-md);
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.05);
  overflow: hidden;
```

```
    position: relative;
  }

  .card::before {
    content: '';
    position: absolute;
    left: 0;
    top: 0;
    height: 100%;
    width: 3px;
    background: var(--trevor-orange);
  }

  /* Additional component styles would continue... */
</style>
```

Example Dashboard Implementation

The dashboard should incorporate these design elements while maintaining functionality:

Unset

```
<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">Monday, March 10, 2025</p>
    </div>
    <div class="illustration-container">
      <!-- Trevor Project inspired illustration -->
      <img src="assets/teamwork-illustration.svg" alt="Team
Support" class="header-illustration">
    </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">96.2%</div>
      <div class="metric-target">Target: 95%</div>
    </div>

    <!-- Additional metrics cards -->
  </div>

  <div class="dashboard-grid">
    <!-- Cards would be here -->
  </div>
</div>
```

This design system combines The Trevor Project's vibrant, hopeful visual identity with a professional, calming interface that acknowledges the serious nature of crisis support work while creating an optimistic environment for team leads to work in.

Final Implementation Notes for Developers

To create a flawless, error-free Crisis Support System that works perfectly on the first implementation while incorporating The Trevor Project's brand identity, please follow these comprehensive guidelines:

System Architecture Overview

The Crisis Support System should be built as a Google Sheets-based application with the following key components:

1. **Core Spreadsheet**
    - Multiple sheets for data storage (as defined in configuration)
    - Named ranges for critical data access
    - Protected ranges for configuration and formulas

2. **Apps Script Backend**
    - Modular code organization with clear separation of concerns
    - Error handling throughout all functions
    - Defensive programming to prevent common failure modes
    - Comprehensive logging for troubleshooting
3. **HTML/CSS Frontend**
    - Consistent visual design based on Trevor Project brand guidelines
    - Responsive interfaces that work across devices
    - Accessibility-compliant components
    - Progressive enhancement for graceful degradation

Error Prevention Implementation

To ensure zero errors on first deployment:

1. **Build a Test Environment First**
    - Create a duplicate sheet for development and testing
    - Implement automated test cases for core functionality
    - Use dummy data to simulate various scenarios
    - Test with multiple users simultaneously
2. **Sheet Validation**
    - Implement the `ensureRequiredSheets()` function to run before any operation
    - Verify column headers match expected format
    - Create missing sheets with proper headers automatically
    - Log any structural changes for audit
3. **UI Context Awareness**
    - Always use the `UIHandler` service instead of direct `SpreadsheetApp.getUi()`
    - Implement context detection before UI operations
    - Provide fallback behaviors for background contexts
    - Use toast messages instead of alerts when appropriate
4. **Data Integrity Protection**
    - Implement proper data validation on all input fields
    - Use LockService for critical write operations
    - Create transaction-like patterns for complex operations
    - Add unique identifiers to all major records

5. **Recovery Mechanisms**
    - Implement auto-save for form data
    - Create fallback HTML templates for all critical components
    - Build self-healing functions that can restore system state
    - Maintain detailed audit logs for recovery

Integration with Trevor Project Brand

Ensure the system visually aligns with The Trevor Project while maintaining professional functionality:

1. **Color Implementation**
    - Use Trevor Orange (#FF786E) as the primary accent color
    - Implement calming gradient backgrounds for content areas
    - Use color purposefully to indicate status and actions
    - Ensure all color combinations meet accessibility standards
2. **Illustration Integration**
    - Use the provided illustrations strategically throughout the interface
    - Implement them as SVG for better performance and scaling
    - Position illustrations as supportive elements, not distractions
    - Consider subtle animation effects for engagement
3. **Typography and Space**
    - Implement the Roboto/Poppins font combination throughout
    - Ensure generous whitespace for readability
    - Use consistent text hierarchy across all pages
    - Implement proper line height and paragraph spacing
4. **Component Styling**
    - Create reusable CSS classes for common elements
    - Implement cards with the Trevor-inspired accent styling
    - Use standardized form controls with Trevor styling
    - Create consistent micro-interactions for all interactive elements

Development Process

To ensure the system works perfectly on the first implementation:

1. **Build in Phases**

    - Create the data structure first
    - Implement core server-side functions
    - Build simple UI components
    - Add interactive elements
    - Polish with animations and refinements
2. **Testing Protocol**

    - Unit test each function individually
    - Integration test between components
    - End-to-end test for common workflows
    - Performance test with realistic data volumes
    - User acceptance testing with actual team leads
3. **Documentation**

    - Create comprehensive inline code documentation
    - Build user guides with screenshots
    - Record video tutorials for complex workflows
    - Document all configuration options
    - Create troubleshooting guidelines
4. **Deployment Strategy**

    - Create a detailed deployment checklist
    - Implement feature flags for gradual rollout
    - Plan for monitoring post-deployment
    - Create a rollback plan for critical issues
    - Schedule follow-up improvements

Specific Implementation Details

1. **HTML File Standardization**

    - Create a template system for consistent HTML structure
    - Implement shared CSS and JavaScript includes
    - Use server-side templating for dynamic content
    - Follow the component patterns defined in the documentation
2. **JavaScript Best Practices**

    - Use modern ES6+ features where supported
    - Implement proper error handling in all client-side code
    - Create reusable utility functions
    - Minimize DOM manipulation for better performance
3. **CSS Organization**

    - Implement a CSS custom properties system for theming

    - Create logical component-based CSS organization
    - Use responsive design patterns throughout
    - Implement print stylesheets for reports
4. **Apps Script Optimization**

    - Minimize API calls with batching where possible
    - Implement proper caching for frequently accessed data
    - Use efficient algorithms for data processing
    - Optimize loops and data transformation functions

Final Quality Assurance

Before deployment, ensure these checks are completed:

1. **Functional Verification**

    - All menu items work correctly
    - All forms submit successfully
    - All reports generate accurately
    - All workflows complete as expected
2. **Error Handling Verification**

    - Test with missing sheets/data
    - Verify UI functions fail gracefully
    - Test concurrent editing scenarios
    - Validate recovery from common errors
3. **Visual Consistency Check**

    - Verify all pages follow the Trevor Project design system
    - Check for consistent spacing and alignment
    - Ensure all text is properly formatted
    - Verify illustrations display correctly
4. **Performance Review**

    - Check load times for all components
    - Verify responsiveness with large data sets
    - Test on slower connections
    - Optimize any slow operations

Interactive Sheets Validation System

The Interactive Sheets Validation System provides a user-friendly way to ensure data integrity and system stability, with real-time feedback and guided resolution steps.

Validation Framework Architecture

JavaScript

```
/**
 * 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_STRU
CTURE] = {
      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_STRU
CTURE].fixed++;
        } 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++;
      }

      // Increment resolved count if fixed
      if (checkResult.fixed) {
        this.validationState.issuesResolved++;
      }
    });

    // 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_STRU
CTURE].fixed++;
        } catch (e) {
          checkResult.fixable = false;
          checkResult.message += ` (Error during fix:
${e.message})`;
        }
      }

      // Add the check result

this.validationState.validationResults[this.CATEGORIES.SHEET_STRU
CTURE].checks.push(checkResult);

      // Increment issues count if not passed
      if (!checkResult.passed) {

this.validationState.validationResults[this.CATEGORIES.SHEET_STRU
CTURE].issues++;
        this.validationState.issuesFound++;
      }

      // Increment resolved count if fixed
      if (checkResult.fixed) {
        this.validationState.issuesResolved++;
      }
    });
  },
```

```
  /**
   * Validates data integrity across sheets
   */
  validateDataIntegrity: function(interactive, autoFix) {
    // Initialize results object for this category

this.validationState.validationResults[this.CATEGORIES.DATA_INTEG
RITY] = {
      checks: [],
      issues: 0,
      fixed: 0
    };

    // Update progress indicator if interactive
    if (interactive) {
      this.updateProgressIndicator('Checking data integrity...',
40);
    }

    // Perform data integrity checks specific to each critical
sheet
    this.validateCounselorData(interactive, autoFix);
    this.validateMetricsData(interactive, autoFix);
    this.validateAlertData(interactive, autoFix);
    this.validateTaskData(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_INTEG
RITY].checks.push(checkResult);

    // Increment issues count if not passed
    if (!checkResult.passed) {

this.validationState.validationResults[this.CATEGORIES.DATA_INTEG
RITY].issues++;
      this.validationState.issuesFound++;
    }

    // Check for duplicate emails
    let emailCounts = {};
    let duplicateEmails = [];

    for (let i = 1; i < data.length; i++) {
      const email = data[i][emailIndex];
      if (!email) continue;

      if (!emailCounts[email]) {
        emailCounts[email] = 1;
      } else {
        emailCounts[email]++;
```

```
        duplicateEmails.push(email);
      }
    }

    // Create unique list of duplicate emails
    duplicateEmails = [...new Set(duplicateEmails)];

    // Create check result
    const dupeCheckResult = {
      name: 'Duplicate Counselor Emails',
      passed: duplicateEmails.length === 0,
      level: this.LEVELS.CRITICAL,
      message: duplicateEmails.length === 0
        ? 'No duplicate counselor emails found'
        : `Found ${duplicateEmails.length} duplicate counselor
email(s)`,
      fixable: false, // Manual fix required
      fixed: false,
      fixMessage: 'Review and resolve duplicate email addresses',
      details: `Duplicate emails: ${duplicateEmails.join(', ')}`
    };

    // Add the check result

this.validationState.validationResults[this.CATEGORIES.DATA_INTEG
RITY].checks.push(dupeCheckResult);

    // Increment issues count if not passed
    if (!dupeCheckResult.passed) {

this.validationState.validationResults[this.CATEGORIES.DATA_INTEG
RITY].issues++;
      this.validationState.issuesFound++;
    }
  },
```

```
  /**
   * Validates metrics data consistency
   */
  validateMetricsData: function(interactive, autoFix) {
    const ss = SpreadsheetApp.getActiveSpreadsheet();
    const sheet =
ss.getSheetByName(CRISIS_SUPPORT_CONFIG.SHEETS.CALL_METRICS);

    // 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 numeric values
    const headers = data[0];
    const numericColumns = [
      'Calls Offered',
      'Calls Accepted',
      'Talk Time (min)',
      'After Call Work (min)',
      'On Queue %'
    ];

    // Get indices for numeric columns
    const numericIndices = numericColumns.map(col =>
headers.indexOf(col)).filter(idx => idx !== -1);

    // Check each row for non-numeric values in numeric columns
    let nonNumericIssues = [];
```

```
    for (let i = 1; i < data.length; i++) {
      numericIndices.forEach(colIdx => {
        const value = data[i][colIdx];
        if (value !== '' && value !== null && (isNaN(value) ||
typeof value !== 'number')) {
          nonNumericIssues.push({
            row: i + 1,
            column: colIdx + 1,
            header: headers[colIdx],
            value: value
          });
        }
      });
    }

    // Create check result
    const checkResult = {
      name: 'Call Metrics Numeric Values',
      passed: nonNumericIssues.length === 0,
      level: this.LEVELS.WARNING,
      message: nonNumericIssues.length === 0
        ? 'All call metrics data has correct numeric format'
        : `Found ${nonNumericIssues.length} non-numeric values in
call metrics data`,
      fixable: true,
      fixed: false,
      fixMessage: 'Convert non-numeric values to numeric or clear
invalid cells',
      details: nonNumericIssues.map(issue =>
        `Row ${issue.row}, Column "${issue.header}": Value
"${issue.value}" is not numeric`
      ).join('\n')
    };

    // If there are issues and we should fix them
```

```
    if (!checkResult.passed && (autoFix || (interactive &&
this.shouldFixInteractively(checkResult)))) {
      try {
        // Fix each non-numeric issue
        nonNumericIssues.forEach(issue => {
          // Get the cell
          const cell = sheet.getRange(issue.row, issue.column);

          // Try to convert to number or clear
          const value = cell.getValue();
          const numValue = Number(value);

          if (!isNaN(numValue)) {
            // Can convert to number
            cell.setValue(numValue);
          } else {
            // Cannot convert, clear the cell
            cell.clearContent();
          }
        });

        // Update check result
        checkResult.fixed = true;
        checkResult.message = `Fixed ${nonNumericIssues.length}
non-numeric values in call metrics data`;

this.validationState.validationResults[this.CATEGORIES.DATA_INTEG
RITY].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.DATA_INTEG
RITY].checks.push(checkResult);

    // Increment issues count if not passed
    if (!checkResult.passed) {

this.validationState.validationResults[this.CATEGORIES.DATA_INTEG
RITY].issues++;
      this.validationState.issuesFound++;
    }
  },

  /**
   * Validates alert data
   */
  validateAlertData: function(interactive, autoFix) {
    // Similar to previous validation methods, implement
alert-specific validation
    // ...
  },

  /**
   * Validates task data
   */
  validateTaskData: function(interactive, autoFix) {
    // Similar to previous validation methods, implement
task-specific validation
    // ...
  },

  /**
   * Validates system configuration
   */
  validateConfiguration: function(interactive, autoFix) {
```

```
    // Initialize results object for this category

this.validationState.validationResults[this.CATEGORIES.CONFIGURAT
ION] = {
      checks: [],
      issues: 0,
      fixed: 0
    };

    // Update progress indicator if interactive
    if (interactive) {
      this.updateProgressIndicator('Checking system
configuration...', 60);
    }

    // Check 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.CONFIGURAT
ION].checks.push(checkResult);

    // Increment issues count if not passed
    if (!checkResult.passed) {

this.validationState.validationResults[this.CATEGORIES.CONFIGURAT
ION].issues++;
      this.validationState.issuesFound++;
    }
  },

  /**
   * Validates access permissions
   */
  validatePermissions: function(interactive, autoFix) {
    // Initialize results object for this category

this.validationState.validationResults[this.CATEGORIES.ACCESS_PER
MISSIONS] = {
      checks: [],
      issues: 0,
      fixed: 0
```

```
    };

    // Update progress indicator if interactive
    if (interactive) {
      this.updateProgressIndicator('Checking access
permissions...', 80);
    }

    try {
      const# Crisis Support System Documentation for 988 Lifeline
LGBTQIA+ Services

## System Overview

This documentation provides a comprehensive explanation of the
Crisis Support System developed for the 988 Lifeline Crisis
Services LGBTQIA+ team. The system is built as a Google Sheets
application with Apps Script integration to facilitate crisis
support management, team leadership, and operational reporting.

## Core Functionality

The Crisis Support System provides comprehensive tools for crisis
support operations management including:

1. **Team Management & Counselor Tracking**
   - Adding and monitoring counselor information
   - Status updates and coaching notes
   - 1:1 session documentation
   - Performance tracking

2. **Call Metrics Collection & Reporting**
   - Daily metrics entry and visualization
   - Performance analysis and trends
   - Counselor-specific metrics
   - Report generation for management
```

```
3. **Alert Management**
   - Creating and managing alerts by severity
   - Email notifications for critical alerts
   - Resolution tracking
   - Historical alert reporting

4. **Task Management**
   - Creating and assigning tasks
   - Priority-based organization
   - Completion tracking
   - Asana integration for external task management

5. **Team Lead Tools**
   - Shift initialization and tracking
   - Time tracking across different activities
   - Activity breakdowns and reports
   - Performance dashboards

6. **Schedule Management**
   - Weekly schedule initialization
   - Shift assignments and tracking
   - Counselor availability management
   - Schedule reporting

## System Structure

The system is organized around several key sheets:

- **Counselor Tracking**: Stores counselor information and status
- **Call Metrics**: Records daily call center performance data
- **Alerts**: Tracks system alerts by severity and status
- **Tasks**: Manages tasks, assignments, and completion status
- **Schedule**: Organizes weekly counselor schedules
- **Team Lead Shifts**: Documents team lead shift information
- **Time Logs**: Records time spent on various activities
```

```
- **Coaching Notes**: Stores coaching feedback and follow-ups
- **One-on-One Notes**: Documents 1:1 meetings with counselors
- **Activity Breakdown**: Provides detailed time allocation
analysis
- **Time Summary**: Summarizes time usage patterns by week

## Team Lead Specific Functions

### Shift Management & Coaching
- **Shift Initialization**: Start each shift with structured
goals and priorities
- **Team Leadership**: Manage a team of up to 12 Crisis Workers
with accountability for performance
- **1:1 Coaching Framework**: Conduct biweekly 30-minute sessions
with structured agendas
  - Human check-in (personal wellness)
  - Highlights and wins (reinforcing feedback)
  - Blockers/challenges (identifying growth areas)
  - Biweekly commitments (SMART goals)
  - Counselor reflection
- **Performance Reviews**: Conduct 90-day, mid-year, and annual
performance evaluations

### Time Management & Tracking
- **Activity Time Tracking**: Document time spent across
different activity categories
  - Primary Shift time (direct service)
  - Administrative time
  - Meeting time
  - 1:1 coaching time
  - Other support activities
- **Activity Reflection**: Document successes and challenges with
different approaches
- **Productivity Analysis**: View trends in time allocation to
optimize team lead effectiveness
```

```
- **Counselor Time Management**: Monitor on-queue percentages,
after-call work, and call metrics

### Service Supervision & Quality
- **On-Shift Supervision**: Provide real-time support to
counselors across all teams
- **Metrics Monitoring**: Focus on answer rates (95% target) and
on-queue time
- **Quality Evaluation**: Regular review of counselor performance
against 70+ quality score target
- **Service Analytics**: Track talk time, handle time, and
counselor productivity metrics

### Administrative Responsibilities
- **Timecard Management**: Weekly/biweekly timecard approval for
team members
- **Attendance Tracking**: Monitor planned and unplanned time
off, maintaining below 5% UTO target
- **Documentation Management**: Maintain coaching records in
CultureAmp/Asana
- **Weekly Team Meetings**: Lead 45-minute weekly meetings (20min
program updates, 25min team activities)

### Task & Alert Management
- **Priority Task Handling**: Manage critical issues requiring
immediate intervention
- **Task Assignment & Delegation**: Create and assign tasks with
clear ownership
- **Alert Management**: Create, monitor, and resolve alerts based
on severity level
- **Project Support**: Manage special projects delegated by
managers

## Data Management
```

The system maintains several interconnected data structures aligned with the 988 Team Lead role requirements:

1. **Counselor Performance Data**
   - Personal information and status
   - Quality evaluation scores (targeting 70+ average)
   - Answer rates (targeting 95% goal)
   - Call volumes (5-7 conversations per shift for Digital, 8-10 for Lifeline)
   - Talk times and handle times
   - On-queue percentages
   - Time utilization breakdowns
   - Unplanned time off tracking (targeting below 5%)

2. **Coaching & Development Data**
   - Structured 1:1 meeting documentation
   - SMART goal tracking with biweekly commitments
   - Performance review documentation (90-day, mid-year, annual)
   - Feedback history with resolution tracking
   - Development plans with specific milestones
   - Coaching notes with behavior observations
   - Policy & protocol adherence tracking

3. **Team Lead Activity Data**
   - Shift initialization records with established goals
   - Time utilization across activity categories
     - Primary Shift time
     - Administrative time
     - Meeting time
     - 1:1 coaching time
     - Other support activities
   - Effectiveness metrics for different approaches
   - Weekly time summaries with pattern analysis
   - Shift summary reports with accomplishments

4. **Service Oversight Data**

```
   - Alert management by severity level
   - Task tracking and completion rates
   - Weekly team meeting notes and action items
   - Quality evaluation results by counselor
   - Service-wide metrics for team performance
   - Call pattern analytics for resource planning
   - Policy adherence monitoring

## Getting Started for Team Leads

1. **Initial Setup and Onboarding**
   - Run the `initializeSystem()` function to create required
sheets
   - Configure email notifications for alerts and critical issues
   - Add your team of counselors (up to 12) to Counselor Tracking
   - Set up individual coaching sections in Asana for each
counselor
   - Configure timecard approval settings in ADP

2. **Daily Shift Management**
   - Initialize your shift using "Team Lead Tools > Initialize
Shift"
     - Document specific goals for the shift
     - Set service priorities based on current needs
   - Use Time Tracker to categorize and document activities
     - Track Primary Shift, Admin, Meeting, 1:1, and Other time
     - Document efficiency observations for continuous
improvement
   - Monitor real-time service metrics (answer rates, on-queue
time)
   - Provide on-shift supervision across all counselor teams
   - Create and manage alerts by severity level
   - Update the Attendance Tracker for any counselor absences

3. **Weekly Leadership Activities**
```

```
   - Conduct biweekly 30-minute 1:1s with each counselor
following the structured format
     - Document all coaching in CultureAmp (or Asana for
contractors)
     - Track SMART goal progress and establish new commitments
   - Lead 45-minute weekly team meetings
     - 20 minutes for program updates using Weekly Connection
Slide Deck
     - 25 minutes for team building and engagement activities
   - Approve timecards for all team members
     - Full-time staff biweekly through ADP
     - Contractors weekly through timesheet spreadsheet
   - Review quality evaluation results for all counselors
   - Update counselor performance metrics and identify coaching
opportunities

4. **Monthly Performance Management**
   - Analyze team metrics against targets
     - Answer rates (95% target)
     - Quality scores (70+ average)
     - Call volumes (5-7 Digital, 8-10 Lifeline per shift)
     - Unplanned time off (below 5% target)
   - Identify performance trends requiring intervention
   - Conduct priority coaching conversations for persistent
issues
   - Document performance concerns requiring follow-up
   - Update performance management documentation
   - Generate monthly team performance summary for managers

## Benefits for Team Management

1. **Enhanced Leadership Effectiveness**
   - Structured framework for coaching up to 12 counselors
   - Centralized view of team performance against clear metrics
   - Early identification of performance issues with quality
alerts
```

```
   - Time utilization analytics to optimize team lead activities
   - Comprehensive documentation for performance management

2. **Improved Counselor Development**
   - Structured biweekly 1:1 framework with consistent
documentation
   - SMART goal tracking with measurable commitments
   - Performance visualization with trend analysis
   - Quality evaluation tracking against 70+ score target
   - Progressive coaching documentation for accountability

3. **Efficient Time Management**
   - Activity categorization across Primary Shift, Admin,
Meeting, and 1:1 time
   - Time utilization patterns with effectiveness measurement
   - Automated time tracking with categorized activities
   - Streamlined timecard approval for different employee types
   - Activity reflection tools to identify time optimization
opportunities

4. **Comprehensive Service Management**
   - Real-time monitoring of answer rates against 95% target
   - Counselor productivity tracking (5-7 Digital, 8-10 Lifeline
conversations)
   - Quality evaluation integration with quality score targets
   - Alert management by severity level with resolution tracking
   - Service pattern analysis to guide resource allocation

## Best Practices for Team Leads

1. **Structured Time Management**
   - Initialize shifts with clear goals and expected outcomes
   - Track time across five categories (Primary Shift, Admin,
Meeting, 1:1, Other)
   - Document activity effectiveness to continuously refine
approaches
```

   - Balance time between direct supervision and coaching responsibilities
   - Prioritize critical supervision needs while maintaining coaching cadence
   - Document time allocation patterns to identify optimization opportunities
   - Maintain separate tracking for shift goals vs. achievements

2. **Effective Coaching Implementation**
   - Follow the structured 1:1 agenda for all biweekly 30-minute sessions
     - Human check-in → Highlights/wins → Blockers/challenges → Commitments
   - Document all coaching interactions in CultureAmp (Asana for contractors)
   - Focus on behaviors rather than metrics in coaching conversations
   - Create SMART goals with each counselor (limit to 3 active commitments)
   - Document all priority coaching outside of regular 1:1s with email follow-up
   - Approach coaching with curiosity rather than punitive mindset
   - Focus redirection on specific behaviors that are explicitly documented

3. **Quality-Focused Performance Management**
   - Monitor counselor quality scores against 70+ target average
   - Track answer rates with expectation of 95% target
   - Monitor counselor productivity (5-7 Digital, 8-10 Lifeline conversations per shift)
   - Keep unplanned time off below 5% on quarterly basis
   - Review performance regularly against clear, measurable expectations
   - Document all performance concerns with specific examples

```
   - Maintain progressive coaching documentation with clear
expectations
   - Follow up immediately on safety-related concerns

4. **Service Excellence Implementation**
   - Lead weekly 45-minute team meetings with structured format
     - 20 minutes for program updates using Weekly Connection
slides
     - 25 minutes for team building and skill development
   - Maintain regular on-shift supervision across all counselor
teams
   - Approve timecards weekly (contractors) or biweekly (FT
staff)
   - Create alerts for critical issues with appropriate severity
assignment
   - Document performance trends that impact service quality
   - Monitor counselor status (on-queue, off-queue, documentation
time)
   - Track talk time and handle time trends to identify service
improvements

## Time Management Subsystem

The Time Management subsystem provides comprehensive tracking and
analysis of team lead activities aligned with the 988 Team Lead
role requirements:

1. **Team Lead Activity Tracking**
   - **Activity Categorization**: Track time across five core
categories
     - Primary Shift time (direct service support)
     - Administrative time (emails, documentation, timecards)
     - Meeting time (team meetings, operations meetings)
     - 1:1 coaching time (biweekly 30-minute sessions)
     - Other support activities (projects, skill camps)
```

```
   - **Real-time Activity Logging**: Document activities as they
occur
   - **Shift Goal Setting**: Document specific objectives for
each shift
   - **Activity Notes**: Record what worked well/poorly for
continuous improvement
   - **Activity Breakdown Reports**: View time allocation across
categories
   - **Pattern Analysis**: Identify optimization opportunities in
time usage

2. **Counselor Time Management**
   - **On-Queue Monitoring**: Track percentage of time counselors
are available
   - **Talk Time Analysis**: Monitor average talk time for
counselors
   - **After-Call Work**: Track documentation time after
conversations
   - **Productivity Metrics**: Monitor conversations per shift
against targets
     - Digital staff: 5-7 conversations per shift target
     - Lifeline staff: 8-10 conversations per shift target
   - **Status Analysis**: Break down how counselors allocate
their time
   - **Break Adherence**: Monitor break and lunch timing
compliance

3. **Coaching Time Optimization**
   - **1:1 Preparation Tracker**: Document time spent preparing
for coaching
   - **Coaching Session Timer**: Ensure 30-minute biweekly
sessions stay on track
   - **Coaching Note Efficiency**: Track time spent on
documentation
   - **Priority Coaching Tracking**: Document time spent on
urgent coaching needs
```

```
   - **Goal Setting and Follow-up**: Track time spent on SMART
goal development
   - **Performance Review Preparation**: Document time spent
preparing evaluations

4. **Time Analysis and Reporting**
   - **Weekly Time Summaries**: View aggregated time usage
patterns
   - **Activity Effectiveness Rating**: Score different
approaches for efficiency
   - **Time vs. Impact Analysis**: Correlate time investments
with outcomes
   - **Optimization Recommendations**: Suggest ways to reallocate
time
   - **Shift Comparison**: Compare productivity across different
shifts and days
   - **Team Lead Benchmarking**: Compare time allocation with
other team leads

## Management Support Features

The system specifically supports the 988 Team Lead role
responsibilities with:

1. **Team Performance Analytics**
   - **Performance Dashboard**: Track key metrics for team of up
to 12 counselors
     ```javascript
     // Example counselor productivity analysis
     const counselorAnalysis =
analyzeCounselorProductivity(counselorData);

     /* Sample output:
     Counselor Productivity Analysis
     ==============================
     Counselor: Alex Johnson (Digital)
```
```

```
     Performance Metrics:
     callsPerShift: 5.4 (Target: 6, Adherence: 90.6%)
     answerRate: 92.3% (Target: 95.0%, Adherence: 97.2%)
     qualityScore: 68.5 (Target: 70, Adherence: 97.9%)
     uto: 4.2% (Target: 5.0%, Adherence: 16.0%)

     Coaching Focus Areas:
     - All metrics within acceptable ranges
     */
     ```
   - **Target Adherence**: Track four key metrics against
targets:
     - Answer Rate (95% target)
     - Quality Evaluation (70+ score target)
     - Productivity (5-7 Digital, 8-10 Lifeline conversations per
shift)
     - Unplanned Time Off (below 5% target)
   - **Comparative Analysis**: View performance trends over time
   - **Coaching Focus Identification**: Automatically highlight
areas needing attention

2. **Time Management Optimization**
   - **Activity Analysis**: Detailed breakdown of time allocation
     ```javascript
     // Example time allocation analysis
     const analysis =
analyzeTeamLeadTimeAllocation(sampleTeamLeadData);

     /* Sample output:
     Team Lead Time Allocation Analysis
     ==================================
     Total time tracked: 2400 minutes

     Time by Category:
     Primary Shift: 1020 minutes (42.5%)
```

```
     Administrative: 480 minutes (20%)
     Meetings: 360 minutes (15%)
     1:1 Coaching: 420 minutes (17.5%)
     Other: 120 minutes (5%)

     Recommendations:
     - Consider increasing time on 1:1 Coaching by approximately
7.5%
     */
     ```
   - **Category Distribution**: Track time across Primary Shift,
Admin, Meeting, 1:1, and Other
   - **Optimization Recommendations**: Receive suggestions for
better time allocation
   - **Effectiveness Tracking**: Document which approaches yield
the best results

3. **Structured Coaching Framework**
   - **1:1 Meeting Template**: Five-section agenda structure
     1. Human Check-in
     2. Highlights and Wins
     3. Blockers/Challenge
     4. Biweekly Commitments
     5. Counselor Reflection
   - **Performance Documentation**: Integrated with CultureAmp
and Asana
   - **Goal Setting Tools**: SMART goal development and tracking
   - **Coaching Email Templates**: Standardized follow-up for
priority coaching

4. **Administrative Efficiency**
   - **Timecard Management**: Streamlined approval for both
employee types
     - Full-time staff (biweekly in ADP)
     - Contractors (weekly via timesheet spreadsheet)
```
```

```
   - **Attendance Tracking**: Integrated UTO monitoring with
alert system
   - **Weekly Team Meeting Tools**: Structured 45-minute format
with resources
   - **Report Generation**: Automated reporting for team
performance

## Support Resources Hub

The Support Resources section serves as a centralized hub for
critical documentation and reference materials that team leads
need to perform their roles effectively:

1. **Crisis Protocols Documentation**
   - PDF Storage: Repository for all crisis protocols documents
   - Quick Reference: Emergency procedures and immediate actions
   - Categorized Access: Sort protocols by severity and situation
type
   - Version Control: Clear indicators of protocol version and
last update date
   - Search Functionality: Find specific procedures quickly

2. **Help Resources**
   - System Documentation: Comprehensive guide to using the
Crisis Support System
   - Role Documentation: Clear explanation of 988 Team Lead
responsibilities
   - FAQ Section: Common questions and troubleshooting advice
   - Video Tutorials: Step-by-step guides for complex processes
   - Contact Information: Support escalation path for system
issues

3. **Manager 1:1 Documentation**
   - Meeting Notes: Storage for biweekly manager-team lead 1:1
notes
```

```
   - Action Items: Track progress on tasks assigned during
manager meetings
   - Resource Sharing: Documents shared by managers for team lead
development
   - Goal Tracking: Document and monitor progress on team lead
performance goals
   - Feedback Documentation: Store feedback received from
managers

4. **Professional Development Resources**
   - Training Materials: Access to training documents and courses
   - Skill Development: Resources for improving coaching and
leadership skills
   - Best Practices: Documentation of successful approaches
   - External Resources: Links to valuable external content
   - Continuous Education: Required and recommended learning
materials

# Comprehensive Menu Structure and Implementation Guide

## Overview

This section provides detailed implementation guidance for
developers creating the Crisis Support System for 988 Lifeline
Crisis Services LGBTQIA+. Each menu section, HTML file
requirement, and functional specification is outlined to ensure a
seamless implementation that works flawlessly on the first
deployment.

## Main Menu Structure (`createMainMenu()`)

The main menu should be implemented as follows with specific HTML
files required for each function:

### 1. 📊 Dashboard
- **Function**: `showDashboard()`
```

- **HTML File**: `dashboard.html`
- **Purpose**: Provide overview of key metrics and system status
- **Content**: Team performance graphs, alert counts, active counselor list, quick links
- **Design**: Clean, card-based layout with metrics prominently displayed
- **Interactive Elements**: Refresh button, drill-down capabilities, date range selector

2. 🗂️ Sidebar
- **Function**: `showSidebar()`
- **HTML File**: `sidebar.html`
- **Purpose**: Quick access to common functions
- **Content**: Compact version of main menu options with key metrics
- **Design**: Vertical layout, collapsible sections, minimal padding
- **Interactive Elements**: Quick action buttons, notifications for pending items

3. 👥 Team Management
This submenu handles all counselor management activities:

a. Add Counselor
- **Function**: `showAddCounselorForm()`
- **HTML File**: `counselor-form.html`
- **Purpose**: Add new counselors to the system
- **Form Fields**: Name, email, phone, status, team, start date, focus/goals, notes
- **Validation**: Required fields, email format, duplicate checking
- **Success Handling**: Clear confirmation, option to add another or return

b. Update Status
- **Function**: `showUpdateStatusForm()`

```
- **HTML File**: `status-form.html`
- **Purpose**: Update counselor status (active, inactive, leave, etc.)
- **Form Fields**: Counselor selector, status options, effective date, notes
- **Validation**: Required selections, documentation for certain status changes
- **Success Handling**: Clear confirmation with action taken

#### c. Add Coaching Note
- **Function**: `showCoachingForm()`
- **HTML File**: `coaching-form.html`
- **Purpose**: Document coaching interactions
- **Form Fields**: Counselor selector, coach name, note type, notes, follow-up date
- **Structure**: Match the coaching note structure from documentation
- **Success Handling**: Option to create follow-up task/calendar event

#### d. Counselor 1:1 Notes
- **Function**: `showOneOnOneNotes()`
- **HTML File**: `one-on-one-notes.html`
- **Purpose**: Document biweekly 1:1 meetings with counselors
- **Structure**: Implement the 5-section format exactly as in documentation:
  1. Human Check-in
  2. Highlights and Wins
  3. Blockers/Challenge
  4. Biweekly Commitments (SMART goals)
  5. Counselor Reflection
- **Design**: Clear section dividers, adequate text space, date tracking
- **Storage**: Save to CultureAmp (full-time staff) or Asana (contractors)
```

4. 📞 Call Metrics
This submenu handles performance data collection and reporting:

a. Enter Daily Metrics
- **Function**: `showMetricsForm()`
- **HTML File**: `metrics-form.html`
- **Purpose**: Input daily call center performance data
- **Form Fields**: Date, calls offered, calls accepted, talk time, after-call work, on-queue percentage
- **Validation**: Numerical validation, reasonable ranges, date validation
- **Design**: Clear input fields with proper input types (number, date, etc.)

b. View Reports
- **Function**: `showMetricsReport()`
- **HTML File**: `metrics-report.html`
- **Purpose**: View performance reports and trends
- **Content**: Interactive charts, filterable data tables, date range selection
- **Design**: Multiple visualization options, export functionality
- **Interactive Elements**: Drill-down capabilities, comparison tools

5. 🚨 Alerts
This submenu handles alert creation and management:

a. Create Alert
- **Function**: `showAlertForm()`
- **HTML File**: `alert-form.html`
- **Purpose**: Create new system alerts for issues
- **Form Fields**: Severity selector, message, category, status
- **Validation**: Required fields, character limits for messages
- **Design**: Clear severity indicators, character counter

b. View Active Alerts
- **Function**: `showActiveAlerts()`
- **HTML File**: `active-alerts.html`
- **Purpose**: View and manage existing alerts
- **Content**: Filterable list of alerts with status indicators
- **Actions**: Resolve, escalate, reassign functionality
- **Design**: Color-coding by severity, clear status indicators

6. ✅ Tasks
This submenu manages task creation and tracking:

a. Create Task
- **Function**: `showTaskForm()`
- **HTML File**: `task-form.html`
- **Purpose**: Create new tasks for team members
- **Form Fields**: Task name, description, assignee, due date, priority
- **Validation**: Required fields, due date validation
- **Design**: Clear priority indicators, date picker

b. Create Asana Task
- **Function**: `showAsanaTaskForm()`
- **HTML File**: `asana-task-form.html`
- **Purpose**: Create tasks in external Asana system
- **Form Fields**: Task name, description, assignee, due date, priority, Asana project
- **Integration**: Proper API handling with error management
- **Success Handling**: Confirmation with Asana task link

7. 📅 Schedule
This submenu handles schedule management:

a. Manage Schedule
- **Function**: `showScheduleManager()`
- **HTML File**: `schedule-manager.html`
- **Purpose**: Manage team schedules and shifts

```
- **Content**: Calendar view, shift assignments, time-off
indicators
- **Actions**: Assign shifts, approve time-off, adjust schedules
- **Design**: Clear calendar layout, color-coding by shift type

#### b. Initialize Week
- **Function**: `showInitializeWeekForm()`
- **HTML File**: `initialize-week-form.html`
- **Purpose**: Set up the schedule for a new week
- **Form Fields**: Week start date, shift pattern, template
selection
- **Validation**: Date validation, template completeness
- **Actions**: Create shifts based on template or previous week

### 8. ⏱️ Team Lead Tools
This submenu contains specialized tools for team leads:

#### a. Initialize Shift
- **Function**: `showShiftInitialization()`
- **HTML File**: `shift-initialization.html`
- **Purpose**: Start a team lead shift with goals and priorities
- **Form Fields**:
  - Shift date & time
  - Shift type
  - Goals for the shift (3-5 fields)
  - Quick notes/priorities
- **Design**: Clean form with adequate text space for goals
- **Success Handling**: Confirmation and option to start time
tracker

#### b. Time Tracker
- **Function**: `showTimeTracker()`
- **HTML File**: `time-tracker.html`
- **Purpose**: Track time spent on different activities
- **Content**:
```

```
  - Activity category selection (Primary Shift, Admin, Meeting,
1:1, Other)
  - Timer functionality
  - Activity notes field
  - Previous activities log
- **Design**: Large, easy-to-click buttons for quick category
switching
- **Interactive Elements**: Start/stop timers, quick category
switching
- **Analysis**: Show time distribution with recommendations

### 9. 🆘 Support Resources
This submenu provides access to documentation and resources:

#### a. Crisis Protocols
- **Function**: `showCrisisProtocols()`
- **HTML File**: `crisis-protocols.html`
- **Purpose**: Central repository for crisis response protocols
- **Content**:
  - Categorized list of PDF protocols
  - Search functionality
  - Quick reference guides
  - Version information and update dates
- **Design**: Clean document library with clear categorization
- **File Handling**: PDF viewer integration, download options

#### b. Help Resources
- **Function**: `showHelp()`
- **HTML File**: `help.html`
- **Purpose**: System help and documentation
- **Content**:
  - User guide
  - FAQ section
  - Video tutorials
  - Troubleshooting guide
  - Contact information
```

```
- **Design**: Searchable, categorized help content
- **Interactive Elements**: Step-by-step guides, expandable
sections

#### c. Manager 1:1 Documentation (New)
- **Function**: `showManagerOneOnOnes()`
- **HTML File**: `manager-one-on-ones.html`
- **Purpose**: Store and manage team lead's 1:1s with their
manager
- **Content**:
  - Meeting date and notes
  - Action items and follow-ups
  - Performance feedback
  - Goal tracking
  - Resource links shared by manager
- **Design**: Similar to counselor 1:1 notes but focused on team
lead development
- **Privacy**: Access restricted to the team lead and their
manager

### 10. ⚙️ Settings
- **Function**: `showSettings()`
- **HTML File**: `settings.html`
- **Purpose**: Configure system settings
- **Content**: Email notifications, Asana integration, display
preferences
- **Design**: Clear sections with appropriate input types
- **Validation**: API key format checking, email validation
- **Success Handling**: Save confirmation, testing options for
integrations

# Trevor Project Inspired Design System

Based on the beautiful illustrations and brand colors you've
shared, we should incorporate The Trevor Project's visual
identity into the Crisis Support System while modernizing it for
```

digital use. Below is a comprehensive design system that maintains brand consistency while creating a calming, supportive interface for team leads.

Color Palette

Primary Colors
- **Trevor Orange**: #FF786E (HEX: #FF786E, CMYK: 0,67,51,0, RGB: 255,120,110)
- **Deep Blue**: #001A4E (HEX: #001A4E, CMYK: 100,93,34,43, RGB: 0,26,78)
- **Purple**: #9A3499 (HEX: #9A3499, CMYK: 39,93,0,0, RGB: 155,52,153)
- **Teal**: #137F6A (HEX: #137F6A, CMYK: 86,29,66,11, RGB: 19,127,106)

Secondary Colors
- **Light Blue**: #4F52DE (HEX: #4F52DE, CMYK: 69,0,11,0, RGB: 79,197,222)
- **Soft Yellow**: #FFAD8D (HEX: #FFAD8D, CMYK: 0,45,40,0, RGB: 255,168,141)
- **Lavender**: #B3AE4A (HEX: #B3AE4A, CMYK: 31,27,93,0, RGB: 179,174,74)
- **Light Purple**: #D58AC (HEX: #F54AC, CMYK: 0,97,3,0, RGB: 245,11,139)

Tertiary/Accent Colors
- **Soft Green**: #BAE2CE (HEX: #BAE2CE, CMYK: 27,0,19,0, RGB: 186,226,206)
- **Pale Yellow**: #FFF2DF (HEX: #FFF2DF, CMYK: 0,3,9,0, RGB: 255,242,223)
- **Light Pink**: #FBCBBE (HEX: #FBCBBE, CMYK: 0,23,20,0, RGB: 251,203,190)
- **Soft Lavender**: #D1CFCC (HEX: #D1CFCC, CMYK: 17,11,14,0, RGB: 209,207,204)

```
### Functional Colors
- **Success**: #137F6A (Teal)
- **Warning**: #FFAD8D (Soft Yellow)
- **Error/Alert**: #F54AC (Light Purple)
- **Info**: #4F52DE (Light Blue)

## Gradient Combinations

To create the soothing, optimistic effect mentioned:

1. **Primary Gradient**:
   - From #FF786E (Trevor Orange) to #FBCBBE (Light Pink)
   - Direction: 135deg (top-left to bottom-right)

2. **Calm Gradient**:
   - From #4F52DE (Light Blue) to #BAE2CE (Soft Green)
   - Direction: 135deg (top-left to bottom-right)

3. **Support Gradient**:
   - From #9A3499 (Purple) to #D58AC (Light Purple)
   - Direction: 45deg (bottom-left to top-right)

4. **Background Gradient**:
   - From #FFF2DF (Pale Yellow) to #FFFFFF (White)
   - Direction: 180deg (top to bottom)

## Typography

- **Primary Font**: Roboto
- **Header Font**: Poppins (for a modern, friendly feel)
- **Font Weights**:
  - Headers: 600 (semibold)
  - Subheaders: 500 (medium)
  - Body: 400 (regular)
  - Emphasis: 700 (bold)
```

```
## Illustration Integration

The illustrations you've shared showcase diverse individuals in
supportive scenarios, using bold colors and simplified forms.
These should be integrated throughout the system:

1. **Dashboard**: Use Image 3 (people communicating) as a
background element or welcome graphic
2. **Support Resources**: Use Image 1 (person self-soothing) to
emphasize self-care
3. **Team Management**: Use Image 2 (person with phone) to
represent connectivity
4. **Login/Welcome**: Use all illustrations in a carousel to
represent diversity and support

## UI Component Styling

### Cards
- Soft rounded corners (12px radius)
- Light shadow: 0 4px 8px rgba(0, 0, 0, 0.05)
- White background or very pale gradient
- Orange accent line on the left or top (3px)

### Buttons
- Primary: Trevor Orange gradient with white text
- Secondary: White with Trevor Orange border and text
- Danger: Light Purple with
```