CPMO Initiative Screening Questionnaire - Azure Static Web App Solution

Executive Summary

The CPMO Initiative Screening Questionnaire is a sophisticated web-based form application deployed as an Azure Static Web App. This solution enables government project managers to classify initiatives as Programme, Project, or Business-As-Usual (BAU) through a dynamic, score-based validation system with progressive disclosure of questions based on user responses. This solution is based on original CPMO Excel (initiative sizing matrix, sreening questionnaire) and Word Forms.

Solution Architecture

Deployment Platform

- **Platform**: Azure Static Web App
- **Hosting**: Microsoft Azure Cloud Infrastructure
- **Integration**: Power Automate workflow for form submission processing
- **Storage**: Browser-based (LocalStorage/SessionStorage) for draft persistence

Technology Stack

- **Frontend**: HTML5, CSS3, Vanilla JavaScript
- **Data Format**: JSON for data serialization
- **Libraries**:
- html2pdf.js (v0.10.1)
- jsPDF (v2.5.1)
- html2canvas (v1.4.1)
- **API Integration**: Azure Logic Apps (Power Automate) via REST endpoint

---

Core Features

1. Score-Based Form Validation

The application implements an intelligent classification system that determines whether an initiative is a Programme, Project, or BAU based on user responses across multiple dimensions:

Classification Dimensions:

- **Timing**: Defined vs. undefined start/end dates
- **Scope**: Related projects vs. unrelated tasks
- **Oversight**: Dedicated management vs. line management
- **Risk Management**: Dedicated vs. departmental
- **Budget Control**: Dedicated vs. functional
- **Benefit Tracking**: Formal vs. informal
- **Change Management**: Structured vs. operational

Each dimension contributes to a cumulative score that determines the initiative's classification.

2. Progressive Disclosure (Conditional Form Fields)

The form dynamically shows or hides sections based on: - Classification score thresholds - User selections in previous sections - Initiative complexity indicators

This approach: - Reduces cognitive load on users - Ensures only relevant questions are presented - Streamlines the completion process - Maintains data quality by avoiding irrelevant fields

3. Modern, Accessible Design

Design Elements:

- **Collapsible Sections**: Organized content with expandable/collapsible panels
- **Visual Indicators**: Warning icons for required/critical sections
- **Progress Tracking**: Scroll indicator showing completion percentage
- **Responsive Layout**: Adaptive design for various screen sizes
- **Color-Coded Feedback**: Visual status indicators (green for success, warnings for incomplete)

User Experience Features:

- Clean, professional government-standard interface
- Intuitive navigation structure

- Clear visual hierarchy
- Consistent styling throughout

JavaScript Functions & Purpose

Data Collection & Management

`buildRadioObject()`

Purpose: Central data aggregation function that captures all form field values in real-time.

What it does: - Queries all form inputs (text, radio, date, email, select, checkbox, textarea) - Collects canvas-based signature data as Base64 images - Creates two data objects: - `formData`: Complete dataset including signatures (for submission) - `exportLink`: URL-safe dataset (for generating shareable links) - Generates a query-string URL for form state sharing - Updates the debug display with current form state - Validates canvas signatures (checks if actually drawn vs. blank)

Triggered by: - Any form field change (change/input events) - Page load (DOMContentLoaded + 100ms delay) - Manual invocation from other functions

Technical Details:

```
// Creates objects for both complete data and URL parameters
let formData = {};     // For Power Automate submission
let exportLink = {};   // For URL generation

// Smart signature validation
function isCanvasEmpty(canvas) {
    let ctx = canvas.getContext("2d");
    let pixelData = ctx.getImageData(0, 0, canvas.width, canvas.height).data;
    // Checks if any pixel has opacity > 0
    for (let i = 0; i < pixelData.length; i += 4) {
        if (pixelData[i + 3] !== 0) return false;
    }
    return true;
}
```

Form Submission & Integration

`sendToPowerAutomate(event)`

Purpose: Handles form submission to Azure Logic Apps (Power Automate) workflow.

What it does: 1. Prevents default form submission behavior 2. Validates that form data exists 3. Sends JSON payload to Power Automate endpoint via POST request 4. Handles success/error responses 5. Updates UI to reflect submission status 6. Saves submission to localStorage for confirmation page 7. Redirects to thank-you page after successful submission 8. Resets form and clears session data

API Endpoint:

```
https://prod-28.westeurope.logic.azure.com:443/workflows/
0cf0be27d4504b8aa24c7389183d902c/triggers/manual/paths/invoke
?api-version=2016-06-01&sp=%2Ftriggers%2Fmanual%2Frun&sv=1.0
&sig=8Pj7td5CpvRMxlaDrj1ATtbrywxcvgq_JoqHiJ1kyHo
```

Success Workflow: 1. Display success alert 2. Save to localStorage: `localStorage.setItem("submittedData", JSON.stringify(exportData))` 3. Change footer background to green 4. Disable submit button 5. Reset form fields 6. Clear sessionStorage 7. Display thank you message 8. Redirect after 1.5 seconds

Error Handling: - Network errors: Caught and logged to console - API errors: User-friendly alert messages - Data validation: Prevents submission if no data exists

---

Data Persistence & Recovery

Save/Load Mechanism

The form implements a sophisticated draft-saving system using browser storage.

LocalStorage vs SessionStorage: - **LocalStorage**: Persistent across browser sessions - Stores final submitted data - Used for confirmation/receipt purposes - **SessionStorage**: Temporary per-tab storage - Stores work-in-progress data - Cleared on form submission or manual clear

Functions:

`loadRowsFromStorage(containerId, prefix)`

Purpose: Restores dynamically added form rows (benefits sections).

What it does: - Called on page load for both financial and non-financial benefits sections - Retrieves row data from sessionStorage - Recreates the dynamic form fields with saved values - Maintains row structure and naming conventions

Parameters: - `containerId`: 'formContainer1' (financial) or 'formContainer2' (non-financial) - `prefix`: 'a' or 'b' (used for field naming)

`downloadFile()`

Purpose: Generates a shareable link file for form resumption.

What it does: 1. Uses the `link` global variable (generated by `buildRadioObject()`) 2. Creates a text file containing the URL with all form data as query parameters 3. Triggers browser download 4. Allows users to resume form completion later by opening the link

Use Case: Users can: - Save progress mid-form - Email the link to colleagues - Continue on different devices - Backup form state before submission

---

Data Reset & Cleanup

Clear Data Button Handler

Purpose: Provides controlled data deletion with user confirmation.

What it does: 1. Calls `MyFunction()` (external function - purpose not shown in provided code) 2. Prompts user with confirmation dialog 3. If confirmed: - Clears all localStorage data - Clears all sessionStorage data - Resets all form fields to default values - Unchecks all radio buttons and checkboxes - Resets select dropdowns to first option - Reloads the page for fresh state 4. Displays confirmation alert

Protection Against Accidental Loss: - Requires explicit user confirmation - Provides clear warning message - Irreversible action (no undo)

Event Listeners & Real-Time Updates

Form Change Detection

```
// Monitors all interactive form elements
document.addEventListener("change", (event) => {
    if (event.target.matches("input[type='text'], input[type='radio'],
        input[type='date'], input[type='checkbox'], select, textarea")) {
        buildRadioObject();
    }
});

// Captures keystroke-level changes
document.addEventListener("input", (event) => {
    if (event.target.matches("input[type='text'], input[type='radio'],
        input[type='date'], input[type='checkbox'], select, textarea")) {
        buildRadioObject();
    }
});
```

Why Both `change` and `input`: - `change`: Fires when user commits a value (blur, select change) - `input`: Fires on every keystroke/character entry - Combined: Ensures real-time data capture without missing any updates

Page Load Initialization

```
window.addEventListener("DOMContentLoaded", (event) => {
    setTimeout(buildRadioObject, 100);
});
```

Purpose: - Builds initial data object after DOM is fully loaded - 100ms delay ensures all dynamically populated elements are ready - Critical for URL parameter pre-population (when resuming from saved link)

Data Flow Architecture

1. User Input → Data Capture

```
User types/selects
    ↓
Event listener triggered (change/input)
    ↓
buildRadioObject() called
    ↓
All form fields queried
    ↓
exportData object updated
    ↓
URL generated for sharing
```

2. Form Submission → Power Automate

```
User clicks Submit
    ↓
sendToPowerAutomate() triggered
    ↓
exportData validated
    ↓
HTTP POST to Azure Logic Apps
    ↓
Response received
    ↓
Success: UI updated, data saved, redirect
    ↓
Error: Alert shown, retry allowed
```

3. Save/Resume Workflow

```
User clicks "Save Form"
    ↓
downloadFile() triggered
    ↓
Current URL with query params generated
    ↓
Text file created and downloaded
    ↓
User opens link later
    ↓
Query parameters parsed
    ↓
Form fields pre-populated
    ↓
User continues from where they left off
```

Global Variables & State Management

Key Global Objects

```
let exportData = {};          // Master data object for submission
let exportLinkData = {};      // URL-safe data for sharing
const baseURL = window.location.href.split('?')[0];  // Clean URL
let link = "";                // Generated shareable link
```

Why Global: - Accessible across all functions - Maintains state throughout session - Enables function chaining and data sharing - Facilitates debugging (visible in debug panel)

Form Structure & Sections

1. Project Information

- Project Name (required)
- Completed By (required)
- Email (required)

- Date of Completion

2. Classification Questions (Score-Based)

- **Timing**: Start/end date definition
- **Scope**: Nature of work (related projects vs. tasks)
- **Oversight and Control**: Management requirements
- **Risk**: Risk management approach
- **Budget Control**: Financial oversight
- **Benefits Tracking**: Measurement approach
- **Change Management**: Change control processes

3. Detailed Project Information (Progressive Disclosure)

Shown based on classification score: - Project objectives - Success criteria - Scope (in/out of scope) - Financial benefits (dynamic rows) - Non-financial benefits (dynamic rows) - Legislative change requirements

4. Digital Signatures

- Senior Responsible Officer (Sponsoring Body)
- Senior Responsible Officer (Supplying Body)
- Project Manager
- Canvas-based signature capture
- Save/Clear functionality per signature

Advanced Features

Dynamic Row Management

The form includes repeatable sections for benefits tracking:

Financial Benefits Section (`formContainer1`): - Users can add multiple benefit rows - Each row captures: benefit description, value, measurement method - Data persisted in sessionStorage - Restored on page reload

Non-Financial Benefits Section (`formContainer2`): - Similar structure for qualitative benefits - Separate storage and naming convention - Independent row management

Signature Capture System

Features: - HTML5 Canvas-based drawing - Touch and mouse support - Clear signature functionality - Signature validation (checks if actually drawn) - Base64 encoding for data transmission - Separate save triggers to Power Automate

Functions (referenced but not shown in provided code): - `clearSignature(n, event)`: Clears specific signature canvas - `saveSignature(n, event)`: Converts canvas to Base64 - `sendSignatureToPowerAutomate(event, n)`: Sends signature data separately

---

Integration Points

Power Automate Workflow

Endpoint Type: HTTP Request Trigger

Expected Payload:

```
{
  "initiative-name": "string",
  "completed-by": "string",
  "email": "string",
  "date": "YYYY-MM-DD",
  "dropdown1": "value",
  "dropdown2": "value",
  // ... all other form fields
  "signatureCanvas1": "...",
  "signatureCanvas2": "...",
  "signatureCanvas3": "..."
}
```

Workflow Actions (typical): 1. Parse JSON payload 2. Validate required fields 3. Store in SharePoint/Dataverse 4. Generate PDF report 5. Send email notifications 6. Update project tracking systems

External Resources

SharePoint Integration: - CPMO Hub for templates and guidance - Glossary of terms (linked in navigation)

Email Support: - Direct mailto link to form administrator - Pre-populated subject and body

---

Security Considerations

Data Protection

- **Client-side storage only**: No server-side database in static app
- **HTTPS enforcement**: Azure Static Web Apps default
- **API endpoint security**: Shared Access Signature (SAS) in Power Automate URL
- **No sensitive data persistence**: Passwords/financial details not stored locally

Best Practices Implemented

- Signature validation prevents empty submissions
- Form data validation before API calls
- Error handling for network failures
- User confirmation for destructive actions (clear data)
- Session data cleared after submission

User Experience Optimizations

Performance Features

1. **Lazy loading**: Collapsible sections load content only when expanded
2. **Debounced updates**: buildRadioObject() optimized to minimize excessive calls
3. **Efficient DOM queries**: Uses specific selectors to reduce processing
4. **Minimal dependencies**: Relies on vanilla JS for core functionality

Accessibility Features

1. **Keyboard navigation**: Tab-friendly form structure
2. **Screen reader support**: Proper label associations
3. **Visual indicators**: Warning icons for required sections
4. **Clear feedback**: Success/error messages prominently displayed

Progressive Enhancement

- Works without JavaScript for basic functionality
- Enhanced features (signatures, dynamic sections) layer on top
- Graceful degradation in older browsers

Development & Deployment

Azure Static Web Apps Benefits

1. **Global CDN**: Fast content delivery worldwide
2. **Automatic HTTPS**: Built-in SSL certificate
3. **GitHub/Azure DevOps integration**: CI/CD pipeline
4. **Staging environments**: Pull request previews
5. **Custom domains**: Easy DNS configuration
6. **API routes**: Optional serverless functions support

File Structure

```
/
├── index.html
├── guidance.html
├── thank-you.html
├── css/
│   └── styles.css
├── javascript/
│   ├── export.js (provided code)
│   └── main.js (additional functionality)
└── img/
    └── warning_icon-01.png
```

Maintenance Considerations

- Version number displayed in footer (currently v1.1)
- Debug panel for troubleshooting (hidden in production)
- Modular JavaScript for easy updates
- Comment-friendly code structure

Future Enhancement Opportunities

Potential Improvements

1. **Offline Support**: Service Worker for true offline functionality
2. **Auto-save**: Periodic background saves to prevent data loss
3. **Multi-language**: Internationalization support

4. **Analytics**: Form completion tracking and abandonment analysis
5. **Validation Enhancement**: Real-time field validation with specific error messages
6. **PDF Generation**: Direct client-side PDF export
7. **Attachment Support**: File upload capability for supporting documents
8. **Version Control**: Track form iterations and changes over time

Integration Expansions

1. **Microsoft Teams**: Bot notifications for submissions
2. **Power BI**: Dashboard for submission analytics
3. **SharePoint**: Direct document library integration
4. **Azure AD**: Single sign-on authentication
5. **Dynamics 365**: CRM integration for project tracking

Troubleshooting Guide

Common Issues

Form Data Not Saving: - Check browser localStorage/sessionStorage quotas - Verify JavaScript console for errors - Ensure cookies/storage not blocked

Submission Failing: - Verify Power Automate endpoint is active - Check network connectivity - Review API permissions and SAS token expiry - Confirm all required fields completed

Signatures Not Capturing: - Verify canvas element is visible (not display:none) - Check if signature actually drawn (not just clicked) - Ensure canvas dimensions properly set in CSS

Progress Not Resuming from Link: - Verify URL contains query parameters - Check field name matches between URL and HTML - Ensure `buildRadioObject()` called after page load

---

Conclusion

The CPMO Initiative Screening Questionnaire represents a sophisticated government digital service that combines user-friendly design with robust technical implementation. Its score-based validation system ensures initiatives are properly classified, while progressive disclosure keeps the form manageable for users. The Azure Static Web App deployment provides reliable, scalable hosting with seamless Power Automate integration for workflow automation.

The solution demonstrates modern web development best practices including: - Progressive enhancement - Responsive design - Real-time data synchronization - Graceful error handling - User-centric design patterns

This technical foundation supports the Jersey Government's digital transformation objectives while maintaining high standards for accessibility, security, and user experience.

Technical Contacts

Form Support: p.wrobel@health.gov.jem
CPMO Hub: SharePoint Templates
Version: 1.1
Last Updated: Per deployment date on Azure Static Web App

Appendix: Complete Function Reference

Data Management

- `buildRadioObject()` - Primary data aggregation
- `isCanvasEmpty(canvas)` - Signature validation

Submission & Integration

- `sendToPowerAutomate(event)` - Form submission handler
- `sendSignatureToPowerAutomate(event, n)` - Signature-specific submission

Persistence & Recovery

- `loadRowsFromStorage(containerId, prefix)` - Restore dynamic rows
- `downloadFile()` - Generate shareable link file
- `addRow(containerId, prefix)` - Add benefit rows dynamically

Signature Management

- `clearSignature(n, event)` - Clear signature canvas
- `saveSignature(n, event)` - Save signature to Base64

UI & Navigation

- `MyFunction()` - Utility function (external)
- Collapsible section handlers (in main.js)

- Scroll indicator updater (in main.js)

Event Handlers

- Form change listener (change event)
- Form input listener (input event)
- DOMContentLoaded initializer
- Button click handlers (submit, download, clear)

This documentation covers the technical implementation of the CPMO Initiative Screening Questionnaire as deployed on Azure Static Web Apps. For user guidance and form completion instructions, refer to the CPMO Guidance page within the application.