**Duplication Handling System Documentation**

**Overview**

The duplication handling system prevents users from inadvertently creating duplicate entries by detecting similarities between new/edited entries and existing data. The system provides a user-friendly interface for resolving duplications through a modal dialog with multiple resolution options.

**Components Architecture**

**Core Components**

1. **Duplicate Detection Logic** - useDuplicateCheck.ts
   * Location: /src/components/data-entry/hooks/useDuplicateCheck.ts
   * Purpose: Provides the core duplicate detection algorithm
2. **Duplicate Modal Dialog** - DuplicateHandler.tsx
   * Location: /src/components/data-entry/components/DuplicateHandler.tsx
   * Purpose: UI component that presents duplicate entries and resolution options
3. **Form Integration** - DataInput/index.tsx
   * Location: /src/components/data-entry/forms/DataInput/index.tsx
   * Purpose: Integrates duplicate detection into the form submission flow

**Logical Flow**

**Duplicate Detection Process**

1. User submits a new entry or edited entry
2. System compares entry against existing data using these criteria:
   * Satisfaction and loyalty values must match
   * At least one identifying field (name, email, date) must have matching non-empty values
   * All other fields must either match or be empty in both entries
3. If duplicate is found, display the duplicate modal
4. User chooses one of three options:
   * Skip entry (discard)
   * Edit before adding (return to form)
   * Add anyway (force save)

**"Between Gates" Edit Process**

The system includes a special "between gates" state where:

1. User has submitted an entry that was flagged as duplicate
2. User chooses to edit the entry
3. Entry is not yet saved to the data table
4. User edits and resubmits
5. System checks again for duplicates against all entries
6. Process repeats until entry is either unique or user chooses another option

**Implementation Details**

**Duplicate Detection Logic**

The duplicate detection uses a multi-step approach:

typescript

Copy

// Core detection logic (simplified)

const isDuplicate = (newEntry, existingEntry) => {

// 1. Check if satisfaction and loyalty match

const valuesMatch = (satisfaction and loyalty match)

if (!valuesMatch) return false;

// 2. Check for matching fields with substantive data

const nameMatch = (both empty OR both non-empty and identical)

const emailMatch = (both empty OR both non-empty and identical)

const dateMatch = (both empty OR both non-empty and identical)

// 3. Check if at least one substantive field matches

const hasSubstantiveMatch = (at least one field where both entries have non-empty matching values)

// Entry is a duplicate if all checks pass

return hasSubstantiveMatch && nameMatch && emailMatch && dateMatch;

}

**Component Interactions**

The system components interact as follows:

1. DataInput - Form submission triggers duplicate check
2. useDuplicateCheck - Performs duplicate detection
3. DuplicateHandler - Displays duplicate information and options
4. DataInput - Handles user's resolution choice

**Duplicate Resolution Options**

1. **Skip Entry** (handleDuplicateSkip):
   * Closes the modal
   * Discards the current entry
   * Shows notification
2. **Edit Before Adding** (handleDuplicateEdit):
   * Closes the modal
   * Keeps current form values
   * Allows user to modify entry
   * Will re-check for duplicates on next submission
3. **Add Anyway** (handleDuplicateAdd):
   * Adds the entry despite duplication
   * Closes the modal
   * Shows success notification

**User Experience**

**Duplicate Detection Workflow**

1. User enters data and submits form
2. If duplicate detected, modal appears showing:
   * Existing entry details
   * New entry details
   * Three resolution options
3. User selects desired action
4. System provides feedback via notification

**Between Gates Editing**

When a user chooses "Edit Before Adding":

1. Modal closes but form retains entered values
2. User can modify any fields
3. On resubmission, system performs a fresh duplicate check
4. If another duplicate is found, modal appears again
5. This cycle can continue until entry is unique or user chooses another option

**Common Scenarios**

**Scenario 1: Same Satisfaction/Loyalty, Different Identifiers**

If user enters same satisfaction/loyalty values but with different identifiers (name, email, date), the system will NOT flag as duplicate.

**Scenario 2: Same Identifiers with Substantive Data**

If user enters same name/email with substantive data, the system will flag as duplicate regardless of satisfaction/loyalty values.

**Scenario 3: Multiple Edit Attempts**

If user repeatedly edits an entry after duplicate warnings, each submission is freshly checked against all existing entries.

**Save this document to: /docs/components/data-entry/duplicate-handling-system.md**