KINTO Smart Ops

Multilingual Implementation Plan

Full UI + Master Data Translation (English • Telugu • Hindi)

KINTO Smart Ops - Full Multilingual Implementation Plan

Version: 1.0 **Date:** November 15, 2025 **Scope:** Complete UI + Master Data multilingual support (English, Telugu, Hindi)

Executive Summary

This plan outlines the complete implementation of multilingual support for KINTO Smart Ops, covering both **User Interface (UI)** translation and **Master Data** translation across all 36 screens and 24 workflows.

Languages Supported:

- English (en) - Primary
- Telugu (te) - □□□□□□
- Hindi (hi) - □□□□□

Total Effort Estimate: 60-70 hours

Table of Contents

Architecture Overview

Two-Layer Approach

Layer 1: UI Translation (Static Text)

- All buttons, labels, menus, error messages
- Uses **react-i18next** library
- Translation files stored in JSON format
- Instant language switching without page reload
- User preference stored in browser localStorage

Layer 2: Master Data Translation (Dynamic Data)

- Product names, machine names, material names, etc.
- Stored in database with separate columns per language
- Forms allow entry in all 3 languages
- Display logic shows correct language based on user selection

Technology Stack

Frontend Libraries:

- `react-i18next` - React bindings for i18next
- `i18next` - Core internationalization framework
- `i18next-browser-languagedetector` - Auto-detect user language

Backend:

- No additional libraries needed
- Database schema changes only

Database Schema Changes

Tables Requiring Multilingual Support

1. Products Table

```
// Current
products {
  name: varchar
  description: text
}

// Multilingual
products {
  name: varchar              // English (primary)
  name_te: varchar          // Telugu
  name_hi: varchar          // Hindi
  description: text         // English (primary)
  description_te: text      // Telugu
  description_hi: text      // Hindi
}
```

2. Machines Table

```
machines {
  name: varchar
  name_te: varchar
  name_hi: varchar
  description: text
  description_te: text
  description_hi: text
}
```

3. Raw Material Types Table

```
rawMaterialTypes {
  name: varchar
  name_te: varchar
  name_hi: varchar
  unit: varchar           // "kg", "liter", etc.
  unit_te: varchar       // "□□□□", "□□□□□"
  unit_hi: varchar       // "□□□□", "□□□□"
}
```

4. Spare Parts Table

```
spareParts {
  name: varchar
  name_te: varchar
  name_hi: varchar
  description: text
  description_te: text
  description_hi: text
}
```

5. Vendors Table

```
vendors {
  name: varchar
  name_te: varchar
  name_hi: varchar
  // Address fields can remain single language
}
```

6. Product Categories Table

```
productCategories {
  name: varchar
  name_te: varchar
  name_hi: varchar
}
```

7. Product Types Table

```
productTypes {
  name: varchar
  name_te: varchar
  name_hi: varchar
}
```

8. QA Checklists Table

```
qaChecklists {
  name: varchar
  name_te: varchar
  name_hi: varchar
  // tasks: json array needs special handling
  tasks: json    // Each task object contains {text, text_te, text_hi}
}
```

9. PM Schedules Table

```
pmSchedules {
  name: varchar
  name_te: varchar
  name_hi: varchar
  // tasks: similar multilingual JSON structure
  tasks: json
}
```

Migration Strategy

Option A: Add Columns (Recommended)

- Add new columns (name_te, name_hi, etc.) to existing tables
- Keep existing English data in original columns
- Allows gradual translation
- No disruption to existing data

Option B: JSON Structure

```
// Alternative approach - single column
name: json {
  en: "Product Name",
  te: "□□□□□□□□□ □□□□",
  hi: "□□□□□□□ □□ □□□"
}
```

- More flexible
- Harder to query/search
- Not recommended for performance reasons

Chosen Approach: Option A (separate columns)

UI Implementation

Translation File Structure

```
client/src/locales/
  ├── en/
  │   └── translation.json
  ├── te/
  │   └── translation.json
  └── hi/
      └── translation.json
```

Translation Categories

1. Common Elements (~100 keys)

- Buttons: Save, Cancel, Delete, Edit, Add, Submit, Close
- Labels: Search, Filter, Export, Print, Actions, Status
- Messages: Loading, Success, Error, Confirmation

2. Authentication (~15 keys)

- Login, Logout, Username, Password, Email
- Welcome messages, Error messages

3. Navigation (~40 keys)

- All sidebar menu items
- Dashboard sections
- Breadcrumbs

4. Forms (~200 keys per screen category)

- Field labels
- Placeholders
- Validation messages
- Help text

5. Tables & Lists (~50 keys)

- Column headers
- Pagination
- Sorting labels
- Empty states

6. Notifications (~100 keys)

- Success messages
- Error messages
- Warning messages
- Info messages

Total Translation Keys: ~1,500-2,000

Component Implementation

Language Selector Component

```
// Location: client/src/components/LanguageSelector.tsx
Features:
  • Dropdown with language flags
  • Shows: □□ English | □□□□□□□ | □□□□□□
  • Saves preference to localStorage
  • Refreshes UI instantly
```

i18n Configuration

```
// Location: client/src/i18n.ts
Features:
  • Auto-detect browser language
  • Fallback to English if translation missing
  • Store preference in localStorage
  • Support for RTL (future)
```

Usage in Components

```
import { useTranslation } from 'react-i18next';

function MyComponent() {
  const { t } = useTranslation();

  return (
    {t('common.save')}
  );
}
```

Master Data Implementation

Form Updates

Product Master Form

Current:

```
</code></pre><p>
<strong>Multilingual:
```