Translation API Implementation Summary

Date: December 22, 2024
Service: Content Service - Translation Management System
Project: Mission Engadi Content Platform
Status: ✅ Complete

Table of Contents

Overview

Purpose

Implemented a comprehensive Translation Management system for the Content Service that enables:
- Multi-language content translation
- Translation workflow management (pending → in_progress → completed → reviewed)
- Language-specific content retrieval
- Bulk translation operations
- Integration with existing content endpoints

Supported Languages

- **EN** - English (Default)
- **ES** - Spanish (Español)
- **FR** - French (Français)
- **PT-BR** - Portuguese - Brazil (Português)

Key Features

✅ Complete CRUD operations for translations
✅ Translation status workflow with validation
✅ Language-aware content retrieval
✅ Bulk translation creation
✅ Authorization and access control

✅ Public/authenticated endpoint distinction
✅ Integration with content endpoints

Architecture

Components Implemented

```
content_service/
├── app/
│   ├── core/
│   │   └── languages.py              # NEW: Language utilities & validation
│   ├── services/
│   │   └── translation_service.py    # NEW: Translation business logic
│   ├── api/v1/
│   │   ├── api.py                    # MODIFIED: Added translation routes
│   │   └── endpoints/
│   │       ├── translations.py       # NEW: 9 translation endpoints
│   │       └── content.py            # MODIFIED: Added language parameter
│   ├── models/
│   │   └── translation.py            # EXISTING: Translation model
│   └── schemas/
│       └── translation.py            # EXISTING: Translation schemas
```

Database Schema

Translation Model:
- `id` : UUID (Primary Key)
- `content_id` : UUID (Foreign Key → content.id)
- `language` : String(10) - Language code
- `translated_title` : String(500)
- `translated_body` : Text
- `translated_slug` : String(500)
- `translator_id` : UUID (Optional, Foreign Key to Auth Service)
- `translation_status` : Enum (pending, in_progress, completed, reviewed)
- `created_at` : Timestamp
- `updated_at` : Timestamp

Indexes:
- Unique index on (content_id, language)
- Index on translation_status
- Index on (language, translation_status)

Implementation Details

1. Language Utilities (`app/core/languages.py`)

Functions:
- `validate_language(language: str)` - Validates language codes
- `get_language_name(language: str)` - Returns display name
- `get_missing_languages(available: list)` - Calculates missing translations

- `is_language_supported(language: str)` - Checks support
- `normalize_language_code(language: str)` - Normalizes variations

Constants:
- `SUPPORTED_LANGUAGES = ["en", "es", "fr", "pt-br"]`
- `LANGUAGE_NAMES` - Mapping of codes to display names

2. Translation Service (`app/services/translation_service.py`)

Core Methods:

```
# CRUD Operations
create_translation(db, content_id, translation_data, translator_id)
get_translation(db, translation_id)
get_translations_for_content(db, content_id, status_filter)
get_translation_by_language(db, content_id, language)
update_translation(db, translation_id, update_data, user_id)
delete_translation(db, translation_id, user_id)

# Workflow Management
change_translation_status(db, translation_id, new_status, user_id)
_validate_status_transition(current_status, new_status)

# Bulk Operations
bulk_create_translations(db, content_id, languages, user_id)
get_available_languages(db, content_id)
```

Status Transitions:

```
pending → in_progress, pending
in_progress → completed, pending, in_progress
completed → reviewed, in_progress, pending, completed
reviewed → in_progress, pending, reviewed
```

3. Translation Endpoints (`app/api/v1/endpoints/translations.py`)

Total: **9 Endpoints**

API Endpoints

1. Create Translation

POST `/api/v1/content/{content_id}/translations`

- **Authentication:** Required
- **Authorization:** Authenticated users
- **Request Body:** `TranslationCreate`
- **Response:** `TranslationResponse` (201)
- **Features:**
- Sets translator_id from authenticated user
- Validates language code
- Checks for duplicate translations
- Default status: pending

Example:

```
POST /api/v1/content/123e4567-e89b-12d3-a456-426614174000/translations
{
  "language": "es",
  "translated_title": "Historia de Misión",
  "translated_body": "Contenido traducido...",
  "translated_slug": "historia-mision"
}
```

2. Get All Translations for Content

GET `/api/v1/content/{content_id}/translations`

- **Authentication:** Optional
- **Query Parameters:**
- `status_filter`: Filter by translation status
- **Response:** `List[TranslationResponse]`
- **Access Control:**
- Public: Only completed/reviewed translations
- Authenticated: All translations

3. Get Translation by Language

GET `/api/v1/content/{content_id}/translations/{language}`

- **Authentication:** Optional
- **Path Parameters:**
- `content_id`: Content UUID
- `language`: Language code (en, es, fr, pt-br)
- **Response:** `TranslationResponse`
- **Access Control:**
- Public: Only completed/reviewed translations
- Authenticated: All translations
- **Error:** 404 if not found

4. Get Translation by ID

GET `/api/v1/content/translations/{translation_id}`

- **Authentication:** Optional
- **Response:** `TranslationResponse`
- **Access Control:**
- Public: Only completed/reviewed translations
- Authenticated: All translations

5. Update Translation

PUT `/api/v1/content/translations/{translation_id}`

- **Authentication:** Required
- **Authorization:** Translator or superuser only
- **Request Body:** `TranslationUpdate`

- **Response:** `TranslationResponse`
- **Features:**
- Validates language if updated
- Validates status transitions
- Checks ownership

6. Delete Translation

DELETE `/api/v1/content/translations/{translation_id}`

- **Authentication:** Required
- **Authorization:** Translator or superuser only
- **Response:** Success message (200)
- **Features:**
- Checks ownership
- Hard delete from database

7. Change Translation Status

POST `/api/v1/content/translations/{translation_id}/status`

- **Authentication:** Required
- **Authorization:** Translator or superuser only
- **Query Parameters:**
- `new_status`: New translation status
- **Response:** `TranslationResponse`
- **Features:**
- Validates status transitions
- Checks ownership

Example:

```
POST /api/v1/content/translations/123e4567.../status?new_status=completed
```

8. Get Available Languages

GET `/api/v1/content/{content_id}/languages`

- **Authentication:** Not required
- **Response:**

```json
{
  "available": ["en", "es", "fr"],
  "missing": ["pt-br"]
}
```

9. Create Bulk Translations

POST `/api/v1/content/{content_id}/translations/bulk`

- **Authentication:** Required
- **Query Parameters:**
- `languages`: List of language codes

- **Response:** `List[TranslationResponse]` (201)
- **Features:**
- Creates placeholder translations
- Sets all to pending status
- Skips existing translations

Example:

```
POST /api/v1/content/123e4567.../translations/bulk?lan-
guages=es&languages=fr&languages=pt-br
```

Workflow Management

Translation Status Flow

```
┌───────────┐
│ PENDING   │◄──────────────┐
└─────┬─────┘               │
      │                     │
      ▼                     │
┌───────────────┐           │
│ IN_PROGRESS   │◄──────────┤
└──────┬────────┘           │
       │                    │
       ▼                    │
┌─────────────┐             │
│ COMPLETED   │─────────────┤
└──────┬──────┘             │
       │                    │
       ▼                    │
┌────────────┐              │
│ REVIEWED   │──────────────┘
└────────────┘
```

Workflow Rules

1. **Pending → In Progress**
   - Translator starts working on translation
   - No special permissions needed

2. **In Progress → Completed**
   - Translator finishes translation
   - All fields must be filled

3. **Completed → Reviewed**
   - Typically requires superuser/reviewer
   - Final approval

4. **Reset to Pending**
   - Can be done from any status
   - Allows rework

Validation

- Status transitions are validated before execution
- Invalid transitions return 400 Bad Request with allowed transitions
- Only translator or superuser can change status

Language Support

Content Endpoint Enhancements

1. Get Content by ID with Language

```
GET /api/v1/content/{content_id}?language=es
```

- Returns content with Spanish translation applied
- Falls back to original if translation not available
- Only uses completed/reviewed translations

2. Get Content by Slug (Already Supported)

```
GET /api/v1/content/slug/{slug}?language=en
```

- Language parameter was already supported
- Now integrates with translation system

Translation Application Logic

```python
# In content endpoint
if language and language != content.language:
    translation = get_translation_by_language(content_id, language)
    if translation and translation.status in [COMPLETED, REVIEWED]:
        content.title = translation.translated_title
        content.body = translation.translated_body
        content.slug = translation.translated_slug
        content.language = translation.language
```

Testing

Manual Testing Checklist

Translation CRUD

- ✅ Create translation for content
- ✅ Get translation by ID
- ✅ Get translations for content
- ✅ Get translation by language
- ✅ Update translation
- ✅ Delete translation

Workflow

- ✅ Change status with valid transitions
- ✅ Prevent invalid status transitions
- ✅ Authorization checks for status changes

Language Support

- ✅ Validate supported language codes
- ✅ Reject unsupported languages
- ✅ Get available/missing languages
- ✅ Bulk create translations

Integration

- ✅ Get content with language parameter
- ✅ Translation applied to content response
- ✅ Fallback to original language

Access Control

- ✅ Public endpoints show only completed translations
- ✅ Authenticated users see all statuses
- ✅ Only translator/admin can update
- ✅ Only translator/admin can delete

Test Scenarios

Scenario 1: Create and Manage Translation

```
# 1. Create content
POST /api/v1/content
{"title": "Mission Update", "body": "...", "slug": "mission-update"}

# 2. Create translation
POST /api/v1/content/{id}/translations
{"language": "es", "translated_title": "Actualización", ...}

# 3. Update translation status
POST /api/v1/content/translations/{trans_id}/status?new_status=in_progress
POST /api/v1/content/translations/{trans_id}/status?new_status=completed

# 4. Get translated content
GET /api/v1/content/{id}?language=es
```

Scenario 2: Bulk Translation Creation

```
# 1. Check available languages
GET /api/v1/content/{id}/languages
# Response: {"available": ["en"], "missing": ["es", "fr", "pt-br"]}

# 2. Create bulk translations
POST /api/v1/content/{id}/translations/bulk?languages=es&languages=fr

# 3. Verify creation
GET /api/v1/content/{id}/translations
# Response: List with 2 pending translations
```

Integration

With Content Service

Content Model:
- Already has `translations` relationship
- No schema changes needed

Content Endpoints:
- Enhanced to support language parameter
- Automatically apply translations when requested

With Auth Service

User Context:
- `translator_id` field links to Auth Service users
- Authorization based on user_id from JWT token

API Router Integration

Updated `/app/api/v1/api.py`:

```
api_router.include_router(
    translations.router,
    prefix="/content",
    tags=["translations"],
)
```

URL Patterns:
- `/api/v1/content/{id}/translations` - Content-specific
- `/api/v1/content/translations/{id}` - Direct translation access

Security & Authorization

Authentication Requirements

Endpoint	Auth Required	Public Access
Create Translation	✅ Yes	❌ No
Get Translation	❌ No	✅ Limited*
Update Translation	✅ Yes	❌ No
Delete Translation	✅ Yes	❌ No
Change Status	✅ Yes	❌ No
Get Languages	❌ No	✅ Yes
Bulk Create	✅ Yes	❌ No

*Public access limited to completed/reviewed translations only

Authorization Rules

1. **Create Translation**
   - Any authenticated user can create
   - translator_id set to current user

2. **Update Translation**
   - Only translator (owner) can update
   - Superusers can update any translation

3. **Delete Translation**
   - Only translator (owner) can delete
   - Superusers can delete any translation

4. **Change Status**
   - Only translator (owner) can change
   - Superusers can change any status

Performance Considerations

Database Queries

Optimizations:
- Unique index on (content_id, language) prevents duplicates
- Indexes on status and language for efficient filtering
- Uses `selectinload` for relationships when needed

Query Patterns:

```
# Efficient lookup by content and language
select(Translation).where(
    and_(
        Translation.content_id == content_id,
        Translation.language == language
    )
)

# Status filtering with index
select(Translation).where(
    Translation.translation_status == status
)
```

Caching Opportunities

Future optimization opportunities:
- Cache completed/reviewed translations
- Cache available languages per content
- Cache language validation results

Error Handling

Common Error Responses

400 Bad Request

```json
{
  "detail": "Language 'xx' is not supported. Supported languages: en, es, fr, pt-br"
}
```

403 Forbidden

```json
{
  "detail": "Not authorized to update this translation"
}
```

404 Not Found

```json
{
  "detail": "Translation for language 'es' not found for this content"
}
```

409 Conflict (via 400)

```json
{
  "detail": "Translation for language 'es' already exists for this content"
}
```

Status Transition Error

```
{
  "detail": "Invalid status transition from 'pending' to 'reviewed'. Allowed
transitions: ['in_progress', 'pending']"
}
```

Next Steps

Immediate Tasks

1. **Testing**
   - [ ] Write unit tests for TranslationService
   - [ ] Write integration tests for translation endpoints
   - [ ] Test workflow transitions
   - [ ] Test authorization logic

2. **Documentation**
   - [ ] Update API documentation (OpenAPI/Swagger)
   - [ ] Add usage examples to README
   - [ ] Document translation workflow for translators

3. **Deployment**
   - [ ] Run database migrations (if needed)
   - [ ] Deploy to staging environment
   - [ ] Perform smoke tests
   - [ ] Deploy to production

Future Enhancements

1. **Translation Memory**
   - Store previously translated segments
   - Suggest translations based on history
   - Reduce duplicate translation work

2. **Machine Translation Integration**
   - Integrate with Google Translate API
   - Provide initial draft translations
   - Allow translators to refine

3. **Translation Quality**
   - Add quality scoring system
   - Track translation metrics
   - Reviewer feedback mechanism

4. **Notifications**
   - Notify translators of new content
   - Alert when translations need review
   - Email/webhook notifications

5. **Advanced Features**
   - Translation versioning
   - Side-by-side comparison view

   - Translation comments/notes
   - Glossary/terminology management

6. **Analytics**
   - Translation completion rates
   - Time to translate metrics
   - Most/least translated languages
   - Translator productivity metrics

Code Structure

Files Created

1. `app/core/languages.py` (136 lines)
   - Language validation and utilities
   - Support for EN, ES, FR, PT-BR
   - Normalization functions

2. `app/services/translation_service.py` (376 lines)
   - Translation CRUD operations
   - Workflow management
   - Bulk operations
   - Status transition validation

3. `app/api/v1/endpoints/translations.py` (400+ lines)
   - 9 REST API endpoints
   - Authorization logic
   - Request/response handling

Files Modified

1. `app/api/v1/api.py`
   - Added translation router
   - Configured URL prefixes

2. `app/api/v1/endpoints/content.py`
   - Added language parameter to get_content_by_id
   - Translation application logic

Total Lines of Code

- New code: ~900 lines
- Modified code: ~50 lines
- **Total: ~950 lines**

API Documentation

Swagger/OpenAPI

Access interactive API documentation:
- **Swagger UI:** `http://localhost:8000/docs`
- **ReDoc:** `http://localhost:8000/redoc`

Endpoint Summary

```
POST   /api/v1/content/{content_id}/translations          Create translation
GET    /api/v1/content/{content_id}/translations          List translations
GET    /api/v1/content/{content_id}/translations/{lang}   Get by language
GET    /api/v1/content/translations/{id}                  Get by ID
PUT    /api/v1/content/translations/{id}                  Update translation
DELETE /api/v1/content/translations/{id}                  Delete translation
POST   /api/v1/content/translations/{id}/status           Change status
GET    /api/v1/content/{content_id}/languages             Get available languages
POST   /api/v1/content/{content_id}/translations/bulk     Bulk create
```

Conclusion

Summary

✅ **Complete Translation Management System Implemented**

The Translation API provides a robust, secure, and scalable solution for managing multi-language content in the Mission Engadi platform. Key achievements:

- **9 fully functional API endpoints**
- **Workflow management with status transitions**
- **Support for 4 languages (EN, ES, FR, PT-BR)**
- **Integration with existing content endpoints**
- **Comprehensive authorization and access control**
- **Bulk operations for efficiency**
- **Production-ready error handling**

Technical Excellence

- Clean architecture with separation of concerns
- Type-safe implementation with Pydantic
- Async/await for optimal performance
- Comprehensive validation
- Security-first design
- RESTful API design

Business Value

- Enables global content distribution
- Supports multilingual missions outreach
- Streamlines translation workflow
- Reduces time-to-market for translated content

- Empowers translators with proper tools

Appendix

Environment Variables

No new environment variables required. Uses existing:
- `DATABASE_URL`
- `SECRET_KEY`
- `JWT_SECRET_KEY`

Dependencies

No new dependencies added. Uses existing:
- FastAPI
- SQLAlchemy
- Pydantic
- PostgreSQL

Database Migration

If using Alembic, run:

```
alembic revision --autogenerate -m "Add translation support"
alembic upgrade head
```

Translation model already exists, so no migration needed if table exists.

Document Version: 1.0
Last Updated: December 22, 2024
Author: Mission Engadi Development Team
Review Status: Ready for Review