Search Service - Complete Implementation Summary

🎯 Overview

The **Search Service** is a powerful full-text search microservice for Mission Engadi that provides universal search capabilities across all content types with multi-language support, advanced filtering, and comprehensive search analytics.

Service Details:
- **Port:** 8011
- **Location:** `/home/ubuntu/search_service`
- **Technology:** FastAPI + PostgreSQL + AsyncIO
- **Search Engine:** PostgreSQL Full-Text Search (tsvector + GIN indexes)

✨ Key Features

1. Full-Text Search

- PostgreSQL tsvector for high-performance full-text search
- GIN (Generalized Inverted Index) for optimal search performance
- Automatic search vector generation via database triggers
- Weighted ranking: Title (A), Content (B), Author (C)

2. Multi-Language Support

- English (en)
- Spanish (es)
- French (fr)
- Portuguese (pt)
- Language-specific text search configurations

3. Search Capabilities

- Universal search across all content types
- Content-specific search (articles, partners, projects, social posts, notifications)
- Relevance-based ranking
- Search result highlighting
- Pagination support
- Custom sorting (relevance, date, title)

4. Advanced Filtering

- Document type filtering
- Language filtering
- Author filtering
- Status filtering

- Date range filtering
- Metadata filtering (JSONB queries)

5. Faceted Search

- Dynamic facet generation
- Count results per facet value
- Filter options for:
- Document types
- Languages
- Authors (top 20)
- Status values

6. Autocomplete & Suggestions

- Trigram similarity search (pg_trgm extension)
- Prefix matching
- Popular searches
- User-specific recent searches
- Automatic suggestion tracking

7. Search Analytics

- Query tracking
- Click-through tracking
- Popular queries analysis
- Zero-result queries monitoring
- Performance metrics
- User search history
- Search statistics dashboard

8. Service Integration

- HTTP clients for all microservices:
- Content Service (articles, stories)
- Partners CRM Service (partners)
- Projects Service (projects)
- Social Media Service (posts)
- Notification Service (notifications)
- Bulk document fetching for re-indexing

📊 Database Models (4 Models)

1. SearchIndex

Main table for storing searchable content with full-text search support.

Fields:
- `id` (UUID) - Primary key
- `document_id` (UUID) - Original document ID (indexed)
- `document_type` (Enum) - Type of document (indexed)

- `title` (String[500]) - Document title
- `content` (Text) - Document content
- `language` (String[10]) - Language code (indexed)
- `metadata` (JSONB) - Additional searchable fields (GIN index)
- `search_vector` (TSVECTOR) - Full-text search vector (GIN index)
- `author_id` (UUID) - Author identifier (indexed)
- `author_name` (String[200]) - Author name
- `status` (String[50]) - Document status
- `published_at` (DateTime) - Publication date (indexed)
- `indexed_at` (DateTime) - Indexing timestamp
- `updated_at` (DateTime) - Last update timestamp

Indexes:
- GIN index on `search_vector` (most important)
- Composite index on `document_type` + `language`
- Individual indexes on key fields
- GIN index on `metadata` JSONB

Automatic Trigger:
- Database trigger automatically updates `search_vector` on insert/update
- Combines title, content, and author name with weights

2. SearchQuery

Tracks all search queries for analytics and insights.

Fields:
- `id` (UUID) - Primary key
- `query_text` (String[500]) - Search query (indexed)
- `language` (String[10]) - Query language
- `filters` (JSONB) - Applied filters
- `results_count` (Integer) - Number of results
- `user_id` (UUID) - User who searched (indexed)
- `execution_time` (Float) - Query execution time (ms)
- `clicked_result_id` (UUID) - Clicked result tracking
- `created_at` (DateTime) - Query timestamp (indexed)

Indexes:
- Composite index on `query_text` + `created_at`
- Composite index on `user_id` + `created_at`
- Index on `results_count` for zero-result analysis

3. SearchSuggestion

Stores popular search suggestions for autocomplete.

Fields:
- `id` (UUID) - Primary key
- `suggestion_text` (String[200]) - Suggestion text (unique, indexed)
- `language` (String[10]) - Suggestion language (indexed)
- `usage_count` (Integer) - Usage frequency
- `last_used_at` (DateTime) - Last usage timestamp

- `created_at` (DateTime) - Creation timestamp
- `updated_at` (DateTime) - Last update timestamp

Indexes:
- Trigram index on `suggestion_text` for fuzzy matching
- Composite indexes for ranking by usage

4. IndexJob

Tracks indexing jobs and their status.

Fields:
- `id` (UUID) - Primary key
- `job_type` (Enum) - Job type: full_reindex, incremental, single_document, bulk
- `status` (Enum) - Status: pending, running, completed, failed (indexed)
- `source_service` (String[100]) - Service that triggered indexing
- `documents_processed` (Integer) - Successfully processed
- `documents_failed` (Integer) - Failed documents
- `error_message` (Text) - Error details
- `started_at` (DateTime) - Start time
- `completed_at` (DateTime) - Completion time
- `created_at` (DateTime) - Creation timestamp (indexed)

Indexes:
- Composite index on `status` + `created_at`
- Composite index on `job_type` + `status`

🔧 Service Layers (6 Services)

1. SearchService

Core search functionality using PostgreSQL full-text search.

Methods:
- `search()` - Universal search with filtering and pagination
- `search_by_type()` - Search within specific document type
- `_prepare_search_query()` - Prepare tsquery for PostgreSQL
- `_apply_filters()` - Apply all search filters
- `_apply_sorting()` - Apply relevance/date/title sorting
- `_to_search_result()` - Convert to SearchResult with highlighting
- `_highlight_text()` - Highlight search terms in results

Features:
- Automatic analytics tracking
- Relevance scoring
- Result highlighting
- Multi-language query preparation

2. IndexingService

Handles indexing operations for searchable content.

Methods:
- `index_document()` - Index single document (create/update)
- `bulk_index()` - Bulk index multiple documents
- `update_index()` - Update existing indexed document
- `delete_from_index()` - Remove document from index
- `reindex_all()` - Re-index all documents from services
- `clear_index()` - Clear entire index
- `get_index_stats()` - Get index statistics

Features:
- Automatic job tracking
- Error handling
- Statistics generation

3. AutoCompleteService

Handles autocomplete and search suggestions.

Methods:
- `get_suggestions()` - Get autocomplete suggestions (trigram + prefix)
- `get_popular_searches()` - Get popular search suggestions
- `get_recent_searches()` - Get user's recent searches
- `track_suggestion()` - Track suggestion usage
- `cleanup_suggestions()` - Clean up low-usage suggestions

Features:
- Trigram similarity search
- Prefix matching fallback
- Usage tracking
- Automatic suggestion creation

4. FacetService

Handles faceted search and filtering.

Methods:
- `get_facets()` - Generate facets for search query
- `get_filter_options()` - Get all available filter options
- `count_results()` - Count results for specific facet value
- `_get_document_type_facets()` - Document type facets
- `_get_language_facets()` - Language facets
- `_get_author_facets()` - Author facets (top 20)
- `_get_status_facets()` - Status facets

Features:
- Dynamic facet generation
- Result counting per facet
- Configurable facet fields

5. SearchAnalyticsService

Tracks and analyzes search queries and user behavior.

Methods:
- `track_search()` - Track search query with analytics

- `track_click()` - Track result click-through
- `get_popular_queries()` - Get most popular queries
- `get_zero_result_queries()` - Get queries with no results
- `get_search_stats()` - Get overall search statistics
- `get_performance_metrics()` - Get performance over time
- `get_user_search_history()` - Get user's search history
- `_update_suggestion()` - Update suggestion from query

Features:
- Comprehensive tracking
- Statistical analysis
- Performance monitoring
- User behavior analysis

6. ServiceIntegration

HTTP client for integrating with other microservices.

Methods:
- `fetch_content_articles()` - Fetch articles from Content Service
- `fetch_partners()` - Fetch partners from Partners CRM
- `fetch_projects()` - Fetch projects from Projects Service
- `fetch_social_posts()` - Fetch posts from Social Media Service
- `fetch_notifications()` - Fetch from Notification Service
- `fetch_all_documents()` - Fetch from all services

Features:
- Async HTTP requests
- Error handling
- Timeout management
- Bulk document fetching

🌐 API Endpoints (25 Endpoints)

Search Endpoints (6 endpoints) - Tag: `search`

1. **POST /api/v1/search**
   - Universal search across all content types
   - Query params: query, document_types, language, filters, pagination, sorting
   - Response: SearchResponse with results and metadata

2. **POST /api/v1/search/content**
   - Search articles and stories only
   - Same parameters as universal search

3. **POST /api/v1/search/partners**
   - Search partners only
   - Filter to partner document type

4. **POST /api/v1/search/projects**
   - Search projects only
   - Filter to project document type

5. **POST /api/v1/search/social**
   - Search social media posts only
   - Filter to social_post document type

6. **POST /api/v1/search/notifications**
   - Search notifications only
   - Filter to notification document type

Autocomplete Endpoints (3 endpoints) - Tag: `autocomplete`

1. **GET /api/v1/autocomplete**
   - Get autocomplete suggestions
   - Query params: query, language, limit
   - Uses trigram similarity + prefix matching

2. **GET /api/v1/autocomplete/popular**
   - Get popular search suggestions
   - Query params: language, limit
   - Returns most frequently searched terms

3. **GET /api/v1/autocomplete/recent** 🔒
   - Get user's recent searches
   - Requires authentication
   - Query params: limit

Indexing Endpoints (5 endpoints) - Tag: `indexing`

1. **POST /api/v1/index/document** 🔒
   - Index a single document
   - Requires authentication
   - Body: IndexDocumentRequest

2. **POST /api/v1/index/bulk** 🔒
   - Bulk index multiple documents
   - Requires authentication
   - Body: BulkIndexRequest with documents array

3. **PUT /api/v1/index/{document_id}** 🔒
   - Update an indexed document
   - Requires authentication
   - Path: document_id, Body: IndexDocumentRequest

4. **DELETE /api/v1/index/{document_id}** 🔒
   - Delete document from index
   - Requires authentication
   - Path: document_id

5. **POST /api/v1/index/reindex** 🔒
   - Re-index all documents from all services
   - Requires authentication
   - Query params: source_service (optional)

Analytics Endpoints (4 endpoints) - Tag: `analytics`

1. **GET /api/v1/analytics/queries** 🔒
   - Get search query statistics
   - Requires authentication
   - Query params: days (default: 30)
2. **GET /api/v1/analytics/popular** 🔒
   - Get most popular search queries
   - Requires authentication
   - Query params: limit, days
3. **GET /api/v1/analytics/zero-results** 🔒
   - Get queries that returned zero results
   - Requires authentication
   - Query params: limit, days
4. **GET /api/v1/analytics/performance** 🔒
   - Get search performance metrics over time
   - Requires authentication
   - Query params: days (default: 7)

Management Endpoints (4 endpoints) - Tag: `management`

1. **GET /api/v1/management/status** 🔒
   - Get search index status and statistics
   - Requires authentication
   - Returns: total documents, by type, by language
2. **POST /api/v1/management/optimize** 🔒
   - Optimize search index (VACUUM ANALYZE)
   - Requires authentication
3. **DELETE /api/v1/management/clear** 🔒
   - Clear all documents from index
   - Requires authentication
   - WARNING: Cannot be undone
4. **GET /api/v1/management/jobs** 🔒
   - List indexing jobs
   - Requires authentication
   - Query params: status, limit

Facets Endpoints (3 endpoints) - Tag: `facets`

1. **POST /api/v1/facets**
   - Get facets for a search query
   - Body: FacetRequest
   - Returns: Available filters with counts

2. **GET /api/v1/facets/options**
    - Get all available options for a filter field
    - Query params: field
3. **POST /api/v1/facets/count**
    - Count results for a specific facet value
    - Query params: query, facet_field, facet_value

🔒 = Requires authentication

🗄️ Database Migration

Migration File: `migrations/versions/001_initial_search_service.py`

Includes:
- All 4 tables with proper schemas
- GIN indexes on tsvector and JSONB columns
- Composite indexes for common queries
- Automatic trigger for search_vector updates
- pg_trgm extension for fuzzy search
- All enum types

To Run Migration:

```
# Navigate to service directory
cd /home/ubuntu/search_service

# Install dependencies
pip install -r requirements.txt

# Run migration
alembic upgrade head
```

🔧 Configuration

Service URLs (app/core/config.py):
- AUTH_SERVICE_URL: http://localhost:8002
- CONTENT_SERVICE_URL: http://localhost:8003
- PARTNERS_CRM_SERVICE_URL: http://localhost:8005
- PROJECTS_SERVICE_URL: http://localhost:8006
- SOCIAL_MEDIA_SERVICE_URL: http://localhost:8007
- NOTIFICATION_SERVICE_URL: http://localhost:8008
- ANALYTICS_SERVICE_URL: http://localhost:8009
- AI_SERVICE_URL: http://localhost:8010

Search Settings:
- SEARCH_MAX_RESULTS: 100
- SEARCH_DEFAULT_PAGE_SIZE: 20

- SEARCH_TIMEOUT_SECONDS: 30
- AUTOCOMPLETE_MIN_LENGTH: 2
- AUTOCOMPLETE_MAX_SUGGESTIONS: 10

🚀 Getting Started

1. Setup Environment

```
cd /home/ubuntu/search_service
cp .env.example .env
# Edit .env with your configuration
```

2. Start Dependencies

```
docker-compose up -d
```

3. Run Migrations

```
alembic upgrade head
```

4. Start Service

```
uvicorn app.main:app --reload --port 8011
```

5. Access Documentation

- Swagger UI: http://localhost:8011/api/v1/docs
- ReDoc: http://localhost:8011/api/v1/redoc

📈 Integration with Other Services

Indexing Flow

1. **Content Service** creates/updates an article
2. Content Service sends indexing request to Search Service
3. Search Service indexes the document with full-text search vectors
4. Document becomes immediately searchable

Search Flow

1. **User** submits search query via any frontend
2. Frontend calls Search Service universal search endpoint
3. Search Service:
   - Executes PostgreSQL full-text search
   - Applies filters
   - Ranks results by relevance

   - Highlights search terms
   - Tracks analytics
4. Returns paginated results with metadata

Re-indexing Flow

1. **Admin** triggers re-index via management endpoint
2. Search Service:
   - Creates index job
   - Fetches all documents from all services
   - Bulk indexes all documents
   - Updates job status
3. All content becomes searchable

🎯 Performance Optimizations

Database Level

- **GIN indexes** on tsvector for fast full-text search
- **Composite indexes** for common query patterns
- **JSONB indexing** for metadata queries
- **Connection pooling** for efficient database connections
- **Automatic VACUUM ANALYZE** via management endpoint

Application Level

- **Async/await** throughout the codebase
- **Efficient query building** with SQLAlchemy
- **Result caching** (can be implemented with Redis)
- **Pagination** to limit result sets
- **Query timeout** protection

Search Quality

- **Weighted ranking** (title > content > author)
- **Language-specific** text search configurations
- **Trigram similarity** for fuzzy autocomplete
- **Prefix matching** for fast suggestions

📊 Analytics & Monitoring

Available Metrics

- Total searches
- Unique queries
- Average results per query
- Average execution time
- Zero-result rate
- Popular queries

- Click-through rates
- User search behavior

Use Cases

- **Content Gaps:** Identify zero-result queries to find missing content
- **Performance:** Monitor execution times and optimize slow queries
- **Popular Topics:** Track trending searches to guide content creation
- **User Behavior:** Understand how users search and what they find

---

🔐 Security Features

- **Authentication required** for indexing, analytics, and management endpoints
- **Optional authentication** for search (tracks user-specific data if authenticated)
- **Input validation** via Pydantic schemas
- **SQL injection protection** via SQLAlchemy ORM
- **Rate limiting** (can be implemented at API gateway level)

🧪 Testing

Test Structure

```
tests/
├── unit/              # Unit tests for services
├── integration/       # Integration tests for endpoints
└── conftest.py        # Test fixtures
```

Run Tests

```
pytest
pytest --cov=app      # With coverage
```

📝 API Documentation

Interactive API documentation is automatically generated and available at:
- **Swagger UI:** http://localhost:8011/api/v1/docs
- **ReDoc:** http://localhost:8011/api/v1/redoc

🌟 Key Highlights

Technical Excellence

✅ **PostgreSQL Full-Text Search** with tsvector + GIN indexes
✅ **Multi-language support** (4 languages)

✅ **Async/await** throughout
✅ **Type hints** everywhere
✅ **Comprehensive error handling**
✅ **Automatic database triggers**
✅ **Production-ready code**

Feature Completeness

✅ **25 API endpoints**
✅ **6 service layers**
✅ **4 database models**
✅ **Faceted search**
✅ **Autocomplete**
✅ **Search analytics**
✅ **Service integration**
✅ **Performance monitoring**

Best Practices

✅ **Clean architecture** (models, schemas, services, endpoints)
✅ **Dependency injection**
✅ **Configuration management**
✅ **Alembic migrations**
✅ **Git version control**
✅ **Comprehensive documentation**

📞 Support

For issues or questions:
1. Check API documentation at `/api/v1/docs`
2. Review this summary document
3. Check logs for error details
4. Contact the development team

🎉 Conclusion

The Search Service is a **production-ready, high-performance search microservice** that provides:

- Universal search across all Mission Engadi content
- Advanced full-text search with PostgreSQL
- Multi-language support
- Comprehensive analytics
- Faceted search and filtering
- Autocomplete and suggestions
- Real-time indexing
- Integration with all microservices

Total Implementation:
- 4 Database Models

- 6 Service Layers
- 25 API Endpoints
- 1 Database Migration
- Complete Git Repository

🚀 **Ready for deployment and integration with Mission Engadi platform!**

Generated: December 25, 2024
Service Version: 0.1.0
Port: 8011
Location: `/home/ubuntu/search_service`