Analytics Service Generation Summary

Date: December 24, 2025
Service: Analytics Service
Port: 8009
Location: `/home/ubuntu/analytics_service`

Overview

The Analytics Service has been successfully generated and configured with core Phase 1 functionality. This service will serve as the central analytics and reporting hub for the Mission Engadi platform, aggregating data from all microservices and providing comprehensive insights.

Service Architecture

Existing Services Integration

The Analytics Service integrates with the following existing services:
- **Auth Service** (Port 8002) - User authentication and authorization data
- **Content Service** (Port 8003) - Content metrics and engagement
- **Partners CRM Service** (Port 8005) - Partner data and relationships
- **Projects Service** (Port 8006) - Project progress and impact metrics
- **Social Media Service** (Port 8007) - Social media post metrics
- **Notification Service** (Port 8008) - Notification delivery metrics

Service Purpose

- Aggregate data from all platform services
- Track metrics across the entire platform
- Provide dashboard data and visualizations
- Generate statistics and trends
- Support 3-year data retention policy

Implementation Details

1. Core Database Models

Metric Model (`app/models/metric.py`)

Purpose: Store analytics metrics from all services

Fields:
- `id` (UUID) - Primary key
- `service_name` (Enum) - Service that generated the metric (auth, content, partners_crm, projects, social_media, notification)
- `metric_type` (Enum) - Type of metric (donation, partner, project, beneficiary, social_post, notification, engagement, conversion, revenue)
- `metric_name` (String, 255) - Name of the metric
- `metric_value` (Float) - Numeric value of the metric
- `metric_unit` (String, 50, nullable) - Unit of measurement (USD, count, percentage, etc.)

- `dimensions` (JSONB) - Additional dimensions (partner_type, project_type, channel, etc.)
- `timestamp` (DateTime) - When metric was recorded
- `date` (Date) - Date for daily aggregations
- `meta` (JSONB) - Additional context
- `created_at` (DateTime) - Record creation timestamp

Indexes:
- Primary: `id`
- Single column: `service_name`, `metric_type`, `metric_name`, `timestamp`, `date`
- Composite: `(service_name, metric_type, date)`, `(metric_name, date)`

Use Cases:
- Track donation amounts and frequencies
- Monitor partner engagement
- Measure project progress and impact
- Analyze social media performance
- Track notification delivery and engagement
- Calculate conversion rates

Dashboard Model (`app/models/dashboard.py`)

Purpose: Store dashboard configurations for different user types

Fields:
- `id` (UUID) - Primary key
- `name` (String, 255) - Dashboard name
- `dashboard_type` (Enum) - Type of dashboard (executive, partner, project, social_media, notification, custom)
- `description` (Text, nullable) - Dashboard description
- `config` (JSONB) - Dashboard configuration (widgets, layout, filters)
- `is_default` (Boolean) - Default dashboard for its type
- `is_public` (Boolean) - Public accessibility flag
- `created_by` (UUID) - User ID who created the dashboard
- `created_at` (DateTime) - Record creation timestamp
- `updated_at` (DateTime) - Record update timestamp

Indexes:
- Primary: `id`
- Single column: `name`, `dashboard_type`, `is_default`, `is_public`, `created_by`

Use Cases:
- Executive dashboards for leadership
- Partner-specific dashboards
- Project tracking dashboards
- Social media analytics dashboards
- Notification analytics dashboards
- Custom user-created dashboards

DataSync Model (`app/models/data_sync.py`)

Purpose: Track data synchronization from other services

Fields:
- `id` (UUID) - Primary key

- `service_name` (Enum) - Service being synchronized
- `sync_type` (Enum) - Type of synchronization (full, incremental, manual)
- `status` (Enum) - Current status (pending, running, completed, failed)
- `started_at` (DateTime, nullable) - When synchronization started
- `completed_at` (DateTime, nullable) - When synchronization completed
- `records_processed` (Integer) - Number of records processed
- `records_failed` (Integer) - Number of records that failed
- `last_sync_timestamp` (DateTime, nullable) - Last successful sync timestamp
- `error_message` (Text, nullable) - Error message if sync failed
- `meta` (JSONB) - Additional metadata
- `created_at` (DateTime) - Record creation timestamp
- `updated_at` (DateTime) - Record update timestamp

Indexes:
- Primary: `id`
- Single column: `service_name`, `sync_type`, `status`, `last_sync_timestamp`

Use Cases:
- Track synchronization from Auth Service
- Monitor Content Service data sync
- Track Partners CRM data updates
- Sync project data and metrics
- Aggregate social media metrics
- Collect notification statistics

2. Pydantic Schemas

Metric Schemas (`app/schemas/metric.py`)

- `MetricBase` - Base schema with common fields
- `MetricCreate` - Schema for creating metrics (POST)
- `MetricUpdate` - Schema for updating metrics (PUT/PATCH)
- `MetricResponse` - Schema for metric responses (GET)
- `MetricAggregation` - Schema for aggregated metric data

Dashboard Schemas (`app/schemas/dashboard.py`)

- `DashboardBase` - Base schema with common fields
- `DashboardCreate` - Schema for creating dashboards (POST)
- `DashboardUpdate` - Schema for updating dashboards (PUT/PATCH)
- `DashboardResponse` - Schema for dashboard responses (GET)
- `DashboardWidgetData` - Schema for widget data

DataSync Schemas (`app/schemas/data_sync.py`)

- `DataSyncBase` - Base schema with common fields
- `DataSyncCreate` - Schema for creating data syncs (POST)
- `DataSyncUpdate` - Schema for updating data syncs (PUT/PATCH)
- `DataSyncResponse` - Schema for data sync responses (GET)
- `DataSyncStats` - Schema for sync statistics

3. Database Migration

Migration File: `migrations/versions/2025_12_24_1750_c1fa66036222_add_core_analytics_models_metric_.py`

Migration Contents:
- Creates 6 PostgreSQL enums:
- `service_name_enum`
- `metric_type_enum`
- `dashboard_type_enum`
- `data_sync_service_name_enum`
- `sync_type_enum`
- `sync_status_enum`
- Creates 3 tables with proper indexes:
- `metrics` - 9 indexes (including 2 composite)
- `dashboards` - 6 indexes
- `data_syncs` - 5 indexes
- Supports rollback with `downgrade()` function

To Apply Migration:

```
cd /home/ubuntu/analytics_service
docker-compose up -d  # Start database
alembic upgrade head
```

Technical Features

UUID Primary Keys

All models use UUID primary keys instead of integer IDs for:
- Better distributed system support
- No ID collision across services
- Enhanced security (non-sequential)

JSONB Fields

Models use JSONB for flexible data storage:
- `Metric.dimensions` - Dynamic metric dimensions
- `Metric.meta` - Additional metric context
- `Dashboard.config` - Dashboard configuration
- `DataSync.meta` - Sync metadata

Comprehensive Indexes

Strategic indexes for common query patterns:
- Time-based queries (timestamp, date)
- Service filtering
- Metric type filtering
- Dashboard type filtering
- Sync status tracking

Type Safety

- SQLAlchemy 2.0+ with type hints

- Pydantic schemas for validation
- Python enums for categorical data

Configuration

Environment Variables

Key configuration in `.env`:
- `PROJECT_NAME="Analytics Service"`
- `PORT=8009`
- `DATABASE_URL="postgresql+asyncpg://postgres:postgres@localhost:5432/analytics_service_db"`
- `REDIS_URL="redis://localhost:6379/0"`
- `KAFKA_BOOTSTRAP_SERVERS="localhost:9092"`

CORS Configuration

Configured for multiple origins:
- `http://localhost:3000` (Frontend dev)
- `http://localhost:8000` (API Gateway)
- `https://engadi.org` (Production)

Git Repository

Initial Commit

```
commit e529204
feat: Add Analytics Service with core models (Metric, Dashboard, DataSync)

- Generate Analytics Service from template
- Implement Metric model for tracking analytics data
- Implement Dashboard model for dashboard configurations
- Implement DataSync model for service synchronization
- Create comprehensive Pydantic schemas
- Add Alembic migration with proper indexes
```

Repository Structure

```
analytics_service/
├── app/
│   ├── models/
│   │   ├── metric.py
│   │   ├── dashboard.py
│   │   └── data_sync.py
│   ├── schemas/
│   │   ├── metric.py
│   │   ├── dashboard.py
│   │   └── data_sync.py
│   ├── api/v1/endpoints/
│   ├── core/
│   ├── db/
│   └── services/
├── migrations/
│   └── versions/
│       └── 2025_12_24_1750_c1fa66036222_add_core_analytics_models_metric_.py
├── tests/
├── .env.example
├── alembic.ini
├── docker-compose.yml
├── Dockerfile
├── requirements.txt
└── README.md
```

Phase 1 Scope

Implemented ✅

- [x] Service generation from template
- [x] Core database models (Metric, Dashboard, DataSync)
- [x] Pydantic schemas for all models
- [x] Alembic migration with indexes
- [x] Git repository initialization
- [x] Configuration setup

Next Steps (Phase 1 Continuation)

- [ ] Implement API endpoints (~25 endpoints):
- Metrics endpoints (CRUD, aggregations, statistics)
- Dashboard endpoints (CRUD, widget data)
- DataSync endpoints (CRUD, sync status, statistics)
- [ ] Data aggregation services
- [ ] Service integration clients
- [ ] Background sync workers
- [ ] Dashboard data generators
- [ ] Statistics and trends calculators

API Endpoints (Planned)

Metrics API (~10 endpoints)

- POST `/api/v1/metrics` - Create metric
- GET `/api/v1/metrics` - List metrics
- GET `/api/v1/metrics/{id}` - Get metric
- GET `/api/v1/metrics/aggregations` - Get aggregated metrics
- GET `/api/v1/metrics/statistics` - Get metric statistics
- GET `/api/v1/metrics/trends` - Get metric trends
- GET `/api/v1/metrics/by-service/{service}` - Get service metrics
- GET `/api/v1/metrics/by-type/{type}` - Get metrics by type
- DELETE `/api/v1/metrics/{id}` - Delete metric
- POST `/api/v1/metrics/bulk` - Bulk create metrics

Dashboards API (~8 endpoints)

- POST `/api/v1/dashboards` - Create dashboard
- GET `/api/v1/dashboards` - List dashboards
- GET `/api/v1/dashboards/{id}` - Get dashboard
- PUT `/api/v1/dashboards/{id}` - Update dashboard
- DELETE `/api/v1/dashboards/{id}` - Delete dashboard
- GET `/api/v1/dashboards/{id}/data` - Get dashboard data
- GET `/api/v1/dashboards/default/{type}` - Get default dashboard
- GET `/api/v1/dashboards/public` - List public dashboards

DataSync API (~7 endpoints)

- POST `/api/v1/syncs` - Create sync job
- GET `/api/v1/syncs` - List sync jobs
- GET `/api/v1/syncs/{id}` - Get sync job
- PUT `/api/v1/syncs/{id}` - Update sync job
- POST `/api/v1/syncs/{id}/start` - Start sync
- POST `/api/v1/syncs/{id}/stop` - Stop sync
- GET `/api/v1/syncs/statistics` - Get sync statistics

Development Workflow

Setup

```
cd /home/ubuntu/analytics_service
python -m venv venv
source venv/bin/activate
pip install -r requirements.txt
cp .env.example .env
# Edit .env with your configuration
```

Database Setup

```
docker-compose up -d  # Start PostgreSQL and Redis
alembic upgrade head   # Apply migrations
```

Run Service

```
uvicorn app.main:app --reload --port 8009
```

Access API Documentation

- Swagger UI: http://localhost:8009/api/v1/docs
- ReDoc: http://localhost:8009/api/v1/redoc

Run Tests

```
pytest
```

Data Retention Policy

The Analytics Service will support a 3-year data retention policy:
- Metrics: 3 years of detailed data
- Aggregations: Permanent storage
- Sync logs: 1 year retention
- Dashboard configs: Permanent storage

Performance Considerations

Optimization Strategies

- Composite indexes for common query patterns
- JSONB for flexible but indexed data
- Date-based partitioning (future)
- Materialized views for aggregations (future)
- Redis caching for frequent queries (future)

Scalability

- Async SQLAlchemy for non-blocking DB operations
- Background workers for data synchronization
- Kafka for event streaming
- Horizontal scaling ready

Security

Authentication

- JWT token validation
- Integration with Auth Service
- Role-based access control

Data Protection

- Encrypted connections (TLS)
- Database connection pooling
- Rate limiting (future)
- Input validation via Pydantic

Monitoring and Logging

Logging

- Structured JSON logging
- Log levels: DEBUG, INFO, WARNING, ERROR, CRITICAL
- Request/response logging

Metrics (Future)

- Prometheus metrics export
- Datadog integration
- Custom analytics metrics

Known Issues and Notes

Resolved Issues

1. **Alembic configuration** - Fixed `version_path_separator` parsing error
2. **CORS origins** - Updated to use JSON array format
3. **Reserved keyword** - Renamed `metadata` field to `meta` (SQLAlchemy reserved keyword)

Important Notes

1. The service is generated but not yet deployed
2. Database migrations are created but not applied
3. API endpoints need to be implemented
4. Integration with other services needs to be configured

Success Criteria

Phase 1 (Current) ✅

- [x] Service generated from template
- [x] Core models implemented
- [x] Schemas created
- [x] Migration generated
- [x] Git repository initialized

Phase 1 (Remaining)

- [ ] Essential endpoints implemented (~25)
- [ ] Data aggregation working
- [ ] Service integration configured
- [ ] Tests written and passing
- [ ] Documentation complete

Resources

Documentation

- README.md - Service overview and setup
- CONTRIBUTING.md - Contributing guidelines
- API Docs - Available at /api/v1/docs

Related Services

- Auth Service: http://localhost:8002
- Content Service: http://localhost:8003
- Partners CRM: http://localhost:8005
- Projects: http://localhost:8006
- Social Media: http://localhost:8007
- Notifications: http://localhost:8008

Conclusion

The Analytics Service foundation has been successfully implemented with:
- ✅ 3 core database models
- ✅ 15 Pydantic schemas
- ✅ 1 comprehensive migration
- ✅ 20+ indexes for optimization
- ✅ UUID support
- ✅ JSONB flexibility
- ✅ Type safety
- ✅ Git version control

The service is ready for Phase 1 continuation with API endpoint implementation and service integration.

Generated by: Mission Engadi Dev
Service Template Version: 0.1.0
Last Updated: December 24, 2025