Ooumph Feed Algorithm 1.0

A comprehensive, production-ready Django-based feed algorithm system for the Ooumph social media platform.

Features

- **Dynamic Feed Composition**: Configurable content ratios (personal connections, trending, discovery, etc.)
- **Extensible Content Types**: Support for posts, communities, products with easy extension
- **Redis Caching**: Performance optimization with intelligent caching strategies
- **Modular Scoring Engines**: Pluggable scoring algorithms for different content types
- **Advanced Analytics**: Comprehensive performance monitoring and analytics
- **REST API**: Complete Django REST Framework integration
- **Production Ready**: Docker deployment, comprehensive logging, testing suite

Quick Start

1. Setup Environment

```
# Clone the repository
cd ooumph_feed/

# Install dependencies
pip install -r requirements.txt

# Use simple settings for development
export DJANGO_SETTINGS_MODULE=ooumph_feed.settings_simple

# Setup database
python manage.py migrate
python manage.py createsuperuser

# Start development server
python manage.py runserver
```

2. Test the API

```
# Health check
curl http://localhost:8000/health/

# Get user feed (requires authentication)
curl -u username:password http://localhost:8000/api/feed/

# Get trending content
curl http://localhost:8000/api/trending/

# Get analytics
curl http://localhost:8000/api/analytics/
```

Architecture

Core Components

1. **Feed Algorithm** (`feed_algorithm/`)
   - Core feed generation logic
   - User preferences and composition
   - Caching and performance optimization
2. **Content Management** (`content_management/`)
   - Abstract ContentItem base class
   - Post, Community, Product implementations
   - Extensible content type system
3. **Scoring Engines** (`scoring_engines/`)
   - Modular scoring algorithms
   - Personal connections scoring
   - Interest-based recommendations
   - Trending and discovery algorithms

4. **Analytics** (`analytics/`)
   - Performance monitoring
   - User engagement tracking
   - A/B testing framework

Feed Composition System

The feed algorithm uses a configurable composition system:

```
default_composition = {
    'personal_connections': 0.40,  # 40% - Content from user's
connections
    'interest_based': 0.25,        # 25% - Content based on user
interests
    'trending_content': 0.15,      # 15% - Currently trending
content
    'discovery_content': 0.10,     # 10% - Discovery and
recommendations
    'community_content': 0.05,     # 5% - Community-based content
    'product_content': 0.05        # 5% - Product/marketplace
content
}
```

API Endpoints

Core Feed API

- `GET /api/feed/` - Get personalized user feed
- `GET /api/feed/composition/` - Get current feed composition
- `POST /api/feed/composition/` - Update feed composition
- `GET /api/trending/` - Get trending content
- `GET /api/analytics/` - Get performance analytics

- `GET /health/` - Health check endpoint

Request Examples

```
# Get feed with pagination
curl "http://localhost:8000/api/feed/?limit=20&offset=0"

# Update feed composition
curl -X POST http://localhost:8000/api/feed/composition/ \
  -H "Content-Type: application/json" \
  -d '{
    "composition": {
      "personal_connections": 0.50,
      "interest_based": 0.30,
      "trending_content": 0.20
    }
  }'

# Get trending content for specific time window
curl "http://localhost:8000/api/trending/?
window=24h&type=post&limit=10"
```

Database Models

Core Models

- **UserProfile**: Extended user information and preferences
- **Connection**: User relationships with circle types (inner/outer/universe)
- **FeedComposition**: User-specific feed composition settings
- **ContentItem**: Abstract base for all content types
- **Post**: Social media posts

- **Community**: Community/group content
- **Product**: Marketplace/product content

Analytics Models

- **Engagement**: User interactions with content
- **FeedDebugEvent**: Feed generation debugging and A/B testing
- **UserInteraction**: Detailed user behavior tracking

Caching Models

- **FeedCache**: Cached feed data
- **TrendingCache**: Trending content cache
- **ConnectionCache**: User connection cache

Configuration

Environment Variables

For production, use these environment variables:

```
# Database
DATABASE_URL=postgresql://user:pass@localhost:5432/ooumph_feed

# Redis
REDIS_URL=redis://localhost:6379/0

# Django
DJANGO_SECRET_KEY=your-secret-key
DJANGO_DEBUG=False
DJANGO_ALLOWED_HOSTS=your-domain.com

# Feed Algorithm
FEED_CACHE_TTL=3600
TRENDING_CACHE_TTL=1800
```

Feed Algorithm Settings

```
# Custom settings
FEED_CACHE_TTL = 3600  # Feed cache TTL in seconds
TRENDING_CACHE_TTL = 1800  # Trending cache TTL in seconds
DEFAULT_FEED_COMPOSITION = {
    'personal_connections': 0.40,
    'interest_based': 0.25,
    'trending_content': 0.15,
    'discovery_content': 0.10,
    'community_content': 0.05,
    'product_content': 0.05
}
```

Development

Running Tests

```
# Run all tests
python manage.py test

# Run specific app tests
python manage.py test feed_algorithm
python manage.py test content_management
```

Code Quality

```
# Check code style
flake8 .

# Run type checking
mypy .
```

Production Deployment

Docker Setup

```
# Build and run with Docker Compose
docker-compose up -d

# Run migrations
docker-compose exec web python manage.py migrate

# Create superuser
docker-compose exec web python manage.py createsuperuser
```

Performance Monitoring

The system includes comprehensive performance monitoring:

- Feed generation timing
- Cache hit/miss rates
- Database query optimization
- Redis performance metrics
- User engagement analytics

Extending the System

Adding New Content Types

1. Create a new model inheriting from `ContentItem`:

```
class Event(ContentItem):
    event_date = models.DateTimeField()
    location = models.CharField(max_length=200)
    # ... additional fields
```

1. Register the content type in the system
2. Update scoring algorithms as needed

Adding Custom Scoring Engines

1. Create a new scoring engine:

```
class CustomScoringEngine:
    def calculate_score(self, content_item, user_profile):
        # Custom scoring logic
        return score
```

1. Register it in the scoring system
2. Update feed composition to include the new factor

Support

For questions and support:

- Check the API documentation at `/admin/`
- Review logs for debugging information
- Monitor performance via `/api/analytics/`

License

Production-ready Django application for Ooumph social media platform.