Technical Documentation: Outlook Recommendation Engine

Project Overview

This is a content recommendation engine designed for news/magazine websites (specifically Outlook India). The system tracks user reading behavior, analyzes article content using NLP techniques, and provides personalized article recommendations based on similarity scores.

System Architecture

Technology Stack

Backend:

- Node.js with Express.js
- MySQL Database
- Python (for NLP processing)

Frontend:

- Vanilla JavaScript
- HTML/CSS

Key Libraries:

- **Python:** NLTK, scikit-learn, spaCy, mysql-connector
- **Node.js:** Express, MySQL, body-parser, cookie-parser, UUID

Core Components

1. Database Schema

The system uses a MySQL database `outlookrecengine` with the following main tables:

- **organizations:** Stores client organization details
- **articles:** Stores article content and metadata
- **related_stories:** Stores similarity scores between articles
- **users:** Tracks user behavior and reading patterns

2. Backend Server (app.js)

Port: 3000

Key Endpoints

GET `/`

- Manages user identification via cookies
- Creates new user ID (UUID) if not present
- Returns user status (new/existing)

GET `/data`

- **Parameters:** `clientID` (query parameter)
- **Purpose:** Serves dynamically generated JavaScript for client websites
- Validates organization by `orgKey`
- Injects user tracking code (`cookies.js`)
- Replaces placeholders: `@userID` , `@clientID`

POST `/api/insertArticle`

- **Purpose:** Stores new article data
- **Validation:** Checks for duplicate articles by title
- **Fields:**
  - title, description, tag, summary, body
  - publish_date, update_date, author
  - category, subcategory, slug, client_id

GET `/api/getRelatedStories`

- **Purpose:** Generates article recommendations
- **Process:**
  1. Checks if article already has related stories
  2. If not, calculates similarity using:
     - Jaccard Index (tag similarity)
     - TF-IDF + Cosine Similarity (content similarity)
  3. Returns top 3 most similar articles
  4. Stores results in `related_stories` table

Key Functions

```
calculateJaccardIndex(arr1, arr2)
```

- Calculates tag overlap between articles
- Returns percentage similarity

```
compareData(currentBody, searchBody, allResult)
```

- Calls Python script for each article comparison
- Uses TF-IDF and cosine similarity
- Populates results array asynchronously

```
callPythonProcess(body1, body2)
```

- Spawns Python child process
- Executes `test1.py` with article bodies
- Returns similarity score (0-100)

3. Python NLP Module (test1.py)

Purpose: Calculate text similarity between two articles

Process:

1. HTML entity decoding
2. Tokenization (lowercase)
3. Remove stopwords
4. Lemmatization
5. TF-IDF vectorization
6. Cosine similarity calculation

Input: Two article texts (command-line arguments)

Output: Similarity score (0-100)

Key Operations:

```
# Preprocessing pipeline
tokens → remove_stopwords → lemmatize → TF-IDF → cosine_similarity
```

4. Content Classification (content_tags.py)

Purpose: Automated tag generation using Named Entity Recognition (NER)

Process:

1. Fetches unclassified articles (`classified = 0`)
2. Uses spaCy NER to extract entities:
   - ORG (Organizations)
   - PERSON (People)
   - GPE (Geopolitical entities)
   - LOC (Locations)
3. Merges with existing tags
4. Updates database with enhanced tags

Model: `en_core_web_sm` (spaCy English model)

5. Client-Side Tracking (cookies.js)

Purpose: Capture article metadata and user behavior

Collected Data:

- Article title, description, summary, body
- Tags, category, author
- Publication/update dates
- Article slug
- User ID and client ID

Metadata Sources:

- HTML meta tags
- JSON-LD structured data
- DOM elements (title, summary)

Data Transmission:

- POST request to `/api/insertArticle`
- Automatic on page load

Cookie Management:

```
setCookie(key, value, expirationDays)
getCookie(key)
```

6. Organization Registration (connect.php)

Purpose: Register new client organizations

Form Fields:

- Organization name, address, phone
- Email ID (validated)
- Domain/website URL

Process:

1. Generate unique 6-character alphanumeric key
2. Insert into `organizations` table
3. Return organization key for integration

Data Flow

Article Ingestion

```
Website → cookies.js → /api/insertArticle → MySQL → content_tags.py → Enhanced Tags
```

Recommendation Generation

```
Request → /api/getRelatedStories → Check Cache → Calculate Similarity → Store Results → Return Top 3
```

Similarity Calculation

```
Article Text → test1.py → Preprocessing → TF-IDF → Cosine Similarity → Score (0-100)
```

Key Algorithms

1. Jaccard Index (Tag Similarity)

```
J(A,B) = |A ∩ B| / |A ∪ B|
```

- Measures tag overlap between articles
- Fast computation
- Good for categorical data

2. TF-IDF + Cosine Similarity (Content Similarity)

```
TF-IDF(t,d) = tf(t,d) × idf(t)
Cosine Similarity = (A · B) / (||A|| ||B||)
```

- Captures semantic content similarity
- Handles variable-length documents
- Industry-standard approach

Configuration

Database Connection

```
{
  host: 'localhost',
  user: 'root',
  password: '',
  database: 'outlookrecengine',
  connectionLimit: 10
}
```

CORS Settings

```
res.header('Access-Control-Allow-Origin', 'http://localhost:5500');
```

Integration Guide

For Client Websites

1. **Register Organization:** Submit organization form to receive unique `clientID`
2. **Embed Tracking Script:**

```
<script src="http://localhost:3000/data?clientID=YOUR_CLIENT_ID"></script>
```

3. **Required HTML Structure:**
   - Meta tags: `description`, `article:tag`, `article:section`
   - JSON-LD structured data with article body
   - Element with class `story-summary`

Security Considerations

⚠️ **Current Issues:**

- No SQL injection protection (uses string concatenation)
- No authentication/authorization
- Hardcoded database credentials
- Open CORS policy
- No input validation on most endpoints

Recommendations:

- Implement parameterized queries
- Add API authentication (JWT/OAuth)
- Use environment variables for credentials
- Implement rate limiting
- Add input sanitization

Performance Optimization

Current Bottlenecks:

- Synchronous Python process spawning
- No caching mechanism
- Real-time similarity calculations

Improvements:

- Implement Redis caching for similarity scores
- Use async/await consistently
- Pre-calculate similarities in background jobs
- Index frequently queried fields

Testing

Test Files Provided:

- `test.html` - Sample article page for testing
- `index.html` - Integration test page

Manual Testing:

1. Start MySQL server
2. Run `node app.js`
3. Open test HTML files in browser
4. Check console logs and network requests

Deployment Notes

Prerequisites:

- Node.js v12+
- Python 3.7+
- MySQL 5.7+
- NLTK data downloaded
- spaCy model installed (`python -m spacy download en_core_web_sm`)

Environment Setup:

```
npm install
pip install -r requirements.txt  # (needs to be created)
python -m nltk.downloader stopwords wordnet punkt
```

Future Enhancements

1. **Machine Learning:** Train custom models on user interaction data
2. **Real-time Updates:** WebSocket for live recommendations
3. **A/B Testing:** Test different recommendation algorithms
4. **Analytics Dashboard:** Visualize recommendation performance
5. **Multi-language Support:** Extend to non-English content
6. **User Profiles:** Build comprehensive user preference models

API Reference Summary

Endpoint	Method	Purpose
`/`	GET	User identification
`/data?clientID=X`	GET	Serve tracking script
`/api/insertArticle`	POST	Store article data
`/api/getRelatedStories`	GET	Get recommendations
`/orgs`	GET	List organizations

Troubleshooting

Common Issues:

1. **Python script fails:** Check NLTK data installation

2. **No recommendations:** Ensure articles have tags and content
3. **CORS errors:** Verify allowed origins in server config
4. **Database connection fails:** Check MySQL service status

Version: 1.0.0
Developer and Documentation: Dron Dasgupta for Outlook India Pvt. Ltd.