Council Assistant - Logging and Feedback Review Guide

Version: 1.0
Date: May 28, 2025
Application: Kent County Council Records Search

Overview

The Council Assistant application automatically logs user interactions, search patterns, performance metrics, and feedback. This guide explains how to access, review, and analyze these logs for system improvement and user support.

Log File Locations

All log files are stored in the `/logs/` directory within your project:

```
/Users/lgfolder/github/council-assistant/logs/
├── search_queries.jsonl          # All search activity
├── user_feedback.jsonl           # User feedback and bug reports
├── user_interactions.jsonl       # UI interactions (tabs, filters)
├── performance.jsonl              # System performance metrics
├── errors.jsonl                   # Application errors
└── application.log                # General application log
```

---

1. User Feedback Review

Location

- **File:** `logs/user_feedback.jsonl`
- **Format:** JSON Lines (one JSON object per line)

Types of Feedback

1. **General Feedback** - User suggestions and comments
2. **Feature Requests** - New functionality requests
3. **Search Quality** - Feedback on search results
4. **Interface Improvement** - UI/UX suggestions
5. **Bug Reports** - Technical issues and problems
6. **Quick Feedback** - Thumbs up/down ratings

Sample Feedback Entry

```json
{
  "timestamp": "2025-05-28T13:15:23.456789",
  "session_id": "uuid-string",
  "event_type": "user_feedback",
  "feedback_type": "Feature Request",
  "message": "Would love to see export functionality for search results",
  "rating": 4,
  "contact_info": "user@example.com",
  "query_context": "climate change initiatives"
}
```

Reviewing Feedback

Method 1: Command Line (Quick Review)

```bash
# View recent feedback
tail -n 20 logs/user_feedback.jsonl

# Count feedback types
grep -o '"feedback_type":"[^"]*"' logs/user_feedback.jsonl | sort | uniq -c

# Find bug reports
grep "Bug Report" logs/user_feedback.jsonl
```

Method 2: Python Script (Detailed Analysis)

```python
import json
import pandas as pd
from datetime import datetime, timedelta

# Load feedback data
feedback_data = []
with open('logs/user_feedback.jsonl', 'r') as f:
    for line in f:
        feedback_data.append(json.loads(line))

df = pd.DataFrame(feedback_data)
df['timestamp'] = pd.to_datetime(df['timestamp'])

# Recent feedback (last 7 days)
recent = df[df['timestamp'] > datetime.now() - timedelta(days=7)]
print(f"Recent feedback count: {len(recent)}")

# Feedback by type
print("\nFeedback Types:")
print(df['feedback_type'].value_counts())

# Average rating
ratings = df[df['rating'].notna()]
print(f"\nAverage rating: {ratings['rating'].mean():.2f}")

# Contact info provided
with_contact = df[df['contact_info'].notna()]
print(f"\nUsers providing contact info: {len(with_contact)}")
```

---

2. Search Analytics

Location

- **File:** `logs/search_queries.jsonl`
- **Purpose:** Track search patterns and success rates

Key Metrics to Monitor

- **Total searches per day/week**
- **Most popular search terms**
- **Zero-result searches** (indicates content gaps)
- **Search success rate**
- **Tab usage patterns**
- **Average search response time**

Sample Search Entry

```json
{
  "timestamp": "2025-05-28T13:10:15.123456",
  "session_id": "uuid-string",
  "event_type": "search_query",
  "query": "road closures traffic management",
  "tab_name": "Meeting Discussions",
  "results_count": 25,
  "search_time_seconds": 1.2,
  "filters": {"sort_method": "Relevance (default)"},
  "query_length": 32,
  "query_word_count": 4
}
```

Analysis Examples

Popular Search Terms

```python
# Load search data
searches = []
with open('logs/search_queries.jsonl', 'r') as f:
    for line in f:
        searches.append(json.loads(line))

df = pd.DataFrame(searches)

# Most popular queries
popular_queries = df['query'].value_counts().head(20)
print("Top 20 Search Terms:")
print(popular_queries)

# Zero result searches (need attention)
zero_results = df[df['results_count'] == 0]
print(f"\nZero result searches: {len(zero_results)}")
print("Failed queries:")
print(zero_results['query'].value_counts().head(10))
```

Performance Monitoring

```python
# Average search times by tab
performance = df.groupby('tab_name')['search_time_seconds'].agg(['mean', 'max', 'count
print("Search Performance by Tab:")
print(performance)

# Slow searches (> 3 seconds)
slow_searches = df[df['search_time_seconds'] > 3.0]
print(f"\nSlow searches: {len(slow_searches)}")
```

---

3. User Interaction Patterns

Location

- **File:** `logs/user_interactions.jsonl`
- **Purpose:** Understand user behavior and interface usage

Tracked Interactions

- **Tab changes** - Which tabs users prefer
- **Filter usage** - Most used filters
- **Pagination** - How users browse results
- **AI summary requests** - AI feature adoption

Analysis Examples

```python
# Load interaction data
interactions = []
with open('logs/user_interactions.jsonl', 'r') as f:
    for line in f:
        interactions.append(json.loads(line))

df = pd.DataFrame(interactions)

# Tab usage
tab_changes = df[df['interaction_type'] == 'tab_change']
print("Tab Usage:")
print(tab_changes['details'].apply(lambda x: x.get('tab_name')).value_counts())

# Filter usage
filters = df[df['interaction_type'] == 'filter_usage']
print("\nFilter Usage:")
print(filters['details'].apply(lambda x: x.get('filter_type')).value_counts())
```

---

4. Error Monitoring

Location

- **File:** `logs/errors.jsonl`
- **Purpose:** Track and resolve application issues

Common Error Types

- **search_error** - Search functionality failures
- **data_loading_error** - Data access issues
- **ai_analysis_error** - AI processing problems
- **pdf_search_error** - Document search failures
- **agenda_search_error** - Meeting search failures

Error Review Process

```python
# Load error data
errors = []
with open('logs/errors.jsonl', 'r') as f:
    for line in f:
        errors.append(json.loads(line))

df = pd.DataFrame(errors)
df['timestamp'] = pd.to_datetime(df['timestamp'])

# Recent errors (last 24 hours)
recent_errors = df[df['timestamp'] > datetime.now() - timedelta(hours=24)]
print(f"Recent errors: {len(recent_errors)}")

# Error types
print("\nError Types:")
print(df['error_type'].value_counts())

# Critical errors to investigate
critical = df[df['error_type'].isin(['data_loading_error', 'search_error'])]
print(f"\nCritical errors: {len(critical)}")
```

---

5. Performance Monitoring

Location

- **File:** `logs/performance.jsonl`
- **Purpose:** Track system performance and identify bottlenecks

Tracked Operations

- **data_loading** - Initial data load time
- **ai_analysis** - AI processing duration
- **Search operations** - Search response times

Performance Analysis

```python
# Load performance data
performance = []
with open('logs/performance.jsonl', 'r') as f:
    for line in f:
        performance.append(json.loads(line))

df = pd.DataFrame(performance)

# Average operation times
avg_times = df.groupby('operation')['duration_seconds'].agg(['mean', 'max', 'count'])
print("Average Operation Times:")
print(avg_times)

# Slow operations (potential issues)
slow_ops = df[df['duration_seconds'] > 5.0]
print(f"\nSlow operations: {len(slow_ops)}")
```

---

6. Regular Monitoring Tasks

Daily Tasks

1. **Check for new feedback** - Review `user_feedback.jsonl`
2. **Monitor error rates** - Check `errors.jsonl` for issues
3. **Review zero-result searches** - Identify content gaps

Weekly Tasks

1. **Analyze search trends** - Popular queries and patterns
2. **Performance review** - Check for slow operations
3. **User interaction analysis** - Tab and filter usage patterns

Monthly Tasks

1. **Comprehensive feedback review** - Contact users who provided emails
2. **Feature request prioritization** - Based on feedback frequency
3. **System optimization** - Address performance bottlenecks

---

7. Automated Analytics

Using the Built-in Analytics Functions

The logging system includes built-in analytics functions:

```python
from modules.utils.logging_system import logger

# Get search analytics for last 7 days
analytics = logger.get_search_analytics(days=7)
print("Search Analytics:")
print(f"Total searches: {analytics['total_searches']}")
print(f"Unique queries: {analytics['unique_queries']}")
print(f"Success rate: {analytics['search_success_rate']:.2%}")

# Get error summary
error_summary = logger.get_error_summary(days=7)
print(f"\nTotal errors: {error_summary['total_errors']}")
print("Error types:", error_summary['error_types'])
```

---

8. Responding to Feedback

High Priority Items

1. **Bug reports** - Investigate and fix immediately
2. **User contact provided** - Follow up within 24-48 hours
3. **Low ratings (1-2 stars)** - Understand and address issues

Feedback Response Process

1. **Acknowledge receipt** (if contact provided)
2. **Investigate issue** or consider feature request
3. **Implement fixes** or communicate timeline
4. **Follow up** with user after resolution

---

9. Privacy and Data Handling

Data Retention

- **Logs rotate automatically** after 90 days
- **Personal data minimization** - Only essential contact info stored
- **Anonymization** - No personally identifiable info in search logs

GDPR Compliance

- Users provide contact info voluntarily
- Right to deletion upon request
- Data used only for service improvement

---

10. Troubleshooting Common Issues

Log Files Not Being Created

1. Check permissions on `/logs/` directory
2. Verify logging module installation
3. Check application error logs

Missing Search Data

1. Confirm search functionality is working
2. Check for import errors in logging modules
3. Verify OpenAI API connectivity

Performance Issues

1. Monitor `performance.jsonl` for slow operations
2. Check data loading times
3. Review search index health

Contact Information

For questions about the logging system or data analysis:

- **Technical Issues:** Check application logs first
- **Feature Requests:** Submit through the app feedback system
- **Data Analysis:** Use provided Python examples

This documentation covers the logging and feedback system for Kent County Council Records Search v1.0. Update as system evolves.