user_engagement_tracker.py

This Python script simulates a microservice that logs, processes, and stores user interaction data (such as 'play', 'pause', 'seek', 'stop') from a video streaming platform.

Purpose:

- Demonstrate clear, humanized technical documentation in code.
- Enable customer support teams to visualize how user sessions are processed.
- Provide scalable architecture hints for agent readiness tooling and content strategy workflows.

Includes:

- RESTful API (Flask)
- SQLite3 database for simulation
- Token-based authentication
- Reusable utilities and handler architecture
- Developer documentation and SEO-rich comments

"""

```
from flask import Flask, request, jsonify
import sqlite3
import uuid
import datetime
import os
```

```
app = Flask(__name__)
DB_FILE = 'user_events.db'
AUTH_TOKEN = 'secure-token-for-demo-only'  # In real-life, use OAuth or API Gateway


# ------------------------------
# Database Setup
# ------------------------------
def init_db():
    """
    Initializes the SQLite database to store user behavior events.
    This structure helps customer experience engineers and data scientists
    analyze real-time interactions for A/B testing or CS escalation resolution.
    """
    conn = sqlite3.connect(DB_FILE)
    cursor = conn.cursor()
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS user_events (
            id TEXT PRIMARY KEY,
            user_id TEXT NOT NULL,
            event_type TEXT NOT NULL,
            timestamp TEXT NOT NULL,
            metadata TEXT
        )
    ''')
```

```
    conn.commit()
    conn.close()

# Initialize DB on first run
if not os.path.exists(DB_FILE):
    init_db()

# ------------------------------
# Authentication Middleware
# ------------------------------
@app.before_request
def authenticate():
    """
    Simple token-based authentication for demo purposes.
    Helps simulate how secure microservices restrict access
    when handling sensitive user interaction logs.
    """
    token = request.headers.get('Authorization')
    if token != f'Bearer {AUTH_TOKEN}':
        return jsonify({'error': 'Unauthorized'}), 401

# ------------------------------
# Utility Functions
# ------------------------------
def save_event_to_db(user_id, event_type, metadata):
```

"""

Saves a structured user event into the database.

Args:

    user_id (str): Unique identifier for the user (e.g., Netflix account ID)

    event_type (str): Type of interaction ('play', 'pause', 'seek', 'stop')

    metadata (str): JSON string containing details like episode, device, etc.

Returns:

    dict: Confirmation message with event ID

"""

conn = sqlite3.connect(DB_FILE)

cursor = conn.cursor()

event_id = str(uuid.uuid4())

timestamp = datetime.datetime.utcnow().isoformat()

cursor.execute('''

    INSERT INTO user_events (id, user_id, event_type, timestamp, metadata)

    VALUES (?, ?, ?, ?, ?)

''', (event_id, user_id, event_type, timestamp, metadata))

conn.commit()

conn.close()

return {'status': 'success', 'event_id': event_id, 'timestamp': timestamp}

```
# ------------------------------
# Routes
# ------------------------------
@app.route('/api/v1/event', methods=['POST'])
def log_event():
    """
    API Endpoint: /api/v1/event [POST]
    Logs a streaming behavior event from a client device.

    Expected JSON Body:
        {
            "user_id": "u12345678",
            "event_type": "play",
            "metadata": {
                "title": "Breaking Bad",
                "season": 3,
                "episode": 7,
                "device": "SmartTV",
                "region": "EMEA"
            }
        }

    Returns:
        JSON confirmation with event_id and timestamp.
    """
```

```
    data = request.get_json()
    if not data or 'user_id' not in data or 'event_type' not in data:
        return jsonify({'error': 'Invalid request payload'}), 400

    response = save_event_to_db(
        user_id=data['user_id'],
        event_type=data['event_type'],
        metadata=str(data.get('metadata', {}))
    )
    return jsonify(response), 201

@app.route('/api/v1/events/<user_id>', methods=['GET'])
def fetch_events(user_id):
    """
    API Endpoint: /api/v1/events/<user_id> [GET]
    Retrieves all logged events for a specific user — helpful for support engineers
    during ticket investigation or playback error analysis.

    Args:
        user_id (str): ID of the user whose events are to be retrieved.

    Returns:
        List of events with timestamps and metadata.
    """
    conn = sqlite3.connect(DB_FILE)
```

```
    cursor = conn.cursor()
    cursor.execute('SELECT * FROM user_events WHERE user_id = ?', (user_id,))
    rows = cursor.fetchall()
    conn.close()

    events = [
        {
            'event_id': row[0],
            'user_id': row[1],
            'event_type': row[2],
            'timestamp': row[3],
            'metadata': row[4]
        }
        for row in rows
    ]
    return jsonify({'events': events})


# ------------------------------
# Launch Server
# ------------------------------
if __name__ == '__main__':
    """
    Launches the Flask development server on port 5000.
    Note: For production, use a WSGI server (Gunicorn, uWSGI) and HTTPS.
    """
```

```
    app.run(debug=True, port=5000)


"""
```

Usage Instructions:

---

1. Start the server: python user_behavior_tracking.py

2. Log a test event with CURL:

```
   curl -X POST http://localhost:5000/api/v1/event \
     -H "Content-Type: application/json" \
     -H "Authorization: Bearer secure-token-for-demo-only" \
     -d '{"user_id": "user_01", "event_type": "play", "metadata": {"title": "Stranger Things", "device": "Laptop"}}'
```

3. Retrieve all events:

```
   curl -X GET http://localhost:5000/api/v1/events/user_01 \
     -H "Authorization: Bearer secure-token-for-demo-only"
```

Bonus:

- Add logging using `logging` module

- Replace SQLite with PostgreSQL for scalability

- Implement OpenAPI schema in a future upgrade

SEO Keywords Embedded:

- python restful api tutorial

- user event logging system in Python

- behavioral analytics streaming platform

- personalization service backend

- customer experience microservices

"""