Comprehensive Developer Guide for Facebook Video Data Tool

Table of Contents

1. Project Overview
2. Architecture
3. Development Environment Setup
4. Code Structure
5. Key Components
6. API Interaction
7. Data Models
8. User Interface
9. Testing Strategy
10. Security Considerations
11. Contribution Guidelines
12. Performance Optimization
13. Future Development

Project Overview

Mission

Develop a user-friendly tool to retrieve and analyze Facebook video performance metrics with minimal technical barriers.

Core Objectives

- Simplify Facebook video data retrieval
- Provide comprehensive performance insights
- Offer flexible data export options
- Maintain cross-platform compatibility

Architecture

Design Principles

- Modular Architecture
- Separation of Concerns
- Extensibility
- Performance Efficiency

Architectural Pattern

Follows a modified Model-View-Controller (MVC) pattern: - **Models**: Data representation and validation - **Views**: User interface components - **Controllers**: Coordination between models and UI

Component Diagram

```
+-------------------+
|   User Interface  |
|   (Tkinter-based) |
+-------------------+
          |
          v
+-------------------+
|   Data Controllers|
|   (Tab Management)|
+-------------------+
          |
    +-----+-----+
    |           |
    v           v
+--------+  +-----------+
| Models |  |    APIs   |
+--------+  +-----------+
    |           |
    v           v
+-------------------+
| Data Transformation|
| & Validation      |
+-------------------+
```

Development Environment Setup

Prerequisites

- Python 3.7+
- pip
- virtualenv (recommended)

Setup Steps

```
# Clone the repository
git clone https://github.com/ericgitonga/utilities.git
cd utilities/fbvideodata

# Create virtual environment
python -m venv venv
source venv/bin/activate  # Windows: venv\Scripts\activate

# Install dependencies
pip install -r requirements.txt
```

```
# Install in editable mode
pip install -e .

# Run the application
python -m fbvideodata.main
```

Code Structure

Directory Layout

```
fbvideodata/

  api/                  # External service interactions
      facebook_api.py
      google_api.py

  models/               # Data models and validation
      video_data.py

  ui/                   # User interface components
      app.py
      setup_tab.py
      data_tab.py
      export_tab.py

  utils/                # Utility functions
      logger.py
      file_utils.py
      update_checker.py

  config.py             # Configuration management
  constants.py          # Application constants
```

Key Components

Facebook API Interaction

- Implements robust API request handling
- Supports pagination
- Error handling and logging
- Version compatibility checks

Example API Request Method

```
def get_page_videos(self, page_id, limit=25, after=None):
    """
    Retrieve videos with robust error handling and pagination
    Args:
        page_id: Facebook page identifier
        limit: Maximum videos to retrieve
        after: Pagination cursor
    """
    fields = [
        "id", "title", "description",
        "created_time", "views"
    ]

    params = {
        "fields": ",".join(fields),
        "limit": limit
    }

    # Pagination support
    if after:
        params["after"] = after

    # Make request with error handling
    result = self._make_request(endpoint, params)
```

Data Models

VideoData Model

- Pydantic-based model for data validation
- Automatic type conversion
- Rich method set for data manipulation

```
class VideoData(BaseModel):
    """Comprehensive video data representation"""
    id: str
    title: Optional[str]
    views: int
    created_time: Optional[datetime]

    @property
    def duration_formatted(self) -> str:
        """Format video duration"""
        minutes = self.length // 60
        seconds = self.length % 60
        return f"{minutes}:{seconds:02d}"
```

User Interface

Design Principles

- Tkinter-based cross-platform UI
- Modular tab-based architecture
- Responsive design
- Error handling in UI components

Tab Components

- Setup Tab: Configuration and connection
- Data Tab: Video data retrieval and display
- Export Tab: Data export options
- Log Tab: Application logging

Testing Strategy

Testing Approach

- Unit Testing
- Integration Testing
- UI Component Testing

Test Coverage Areas

- API interaction
- Data model validation
- Export functionality
- Error handling

Example Test Structure

```python
def test_video_data_model():
    """Test VideoData model validation"""
    video = VideoData(
        id="test_video",
        title="Test Video",
        views=1000
    )
    assert video.id == "test_video"
    assert video.views == 1000
```

Security Considerations

Key Security Practices

- Token encryption
- Secure storage of credentials

- Minimal permission tokens
- Logging with data scrubbing

Token Security Example

```
class SecureTokenManager:
    def encrypt_token(self, token):
        """Securely encrypt access tokens"""
        # Implement encryption logic
        pass

    def decrypt_token(self, encrypted_token):
        """Safely decrypt tokens"""
        # Implement decryption logic
        pass
```

Contribution Guidelines

Contribution Process

1. Fork the repository
2. Create feature branch
3. Implement changes
4. Write/update tests
5. Run tests
6. Submit pull request

Code Style

- Follow PEP 8
- Use type hints
- Write comprehensive docstrings
- Maintain clean, readable code

Review Process

- Automated CI checks
- Code review by maintainers
- Performance and security assessment

Performance Optimization

Strategies

- Implement request caching
- Use efficient data structures
- Minimize API calls
- Optimize UI rendering

Future Development

Potential Enhancements

- Multi-platform support
- Advanced data visualization
- Machine learning insights
- Additional social media platform support

Tools and Libraries

Core Libraries

- `requests`: API interactions
- `pandas`: Data manipulation
- `pydantic`: Data validation
- `tkinter`: User interface
- `gspread`: Google Sheets integration

Documentation

Maintaining Documentation

- Keep README updated
- Maintain comprehensive docstrings
- Update developer and user guides
- Document new features and changes

Conclusion

This project demonstrates a well-structured, modular approach to developing a cross-platform data analysis tool with a focus on user experience, performance, and security.

Note: This guide is a living document. Always refer to the most recent version in the repository.