# PROJECT REPORT

# Project Report: Economic Intelligence Platform

**Author**: KUSHAGRA SINGH BISEN

**SAP ID**: 590014177

**Batch**: BCA-B2

**Repository**: `https://github.com/kushgbisen/news_aggregator`

## Executive Summary

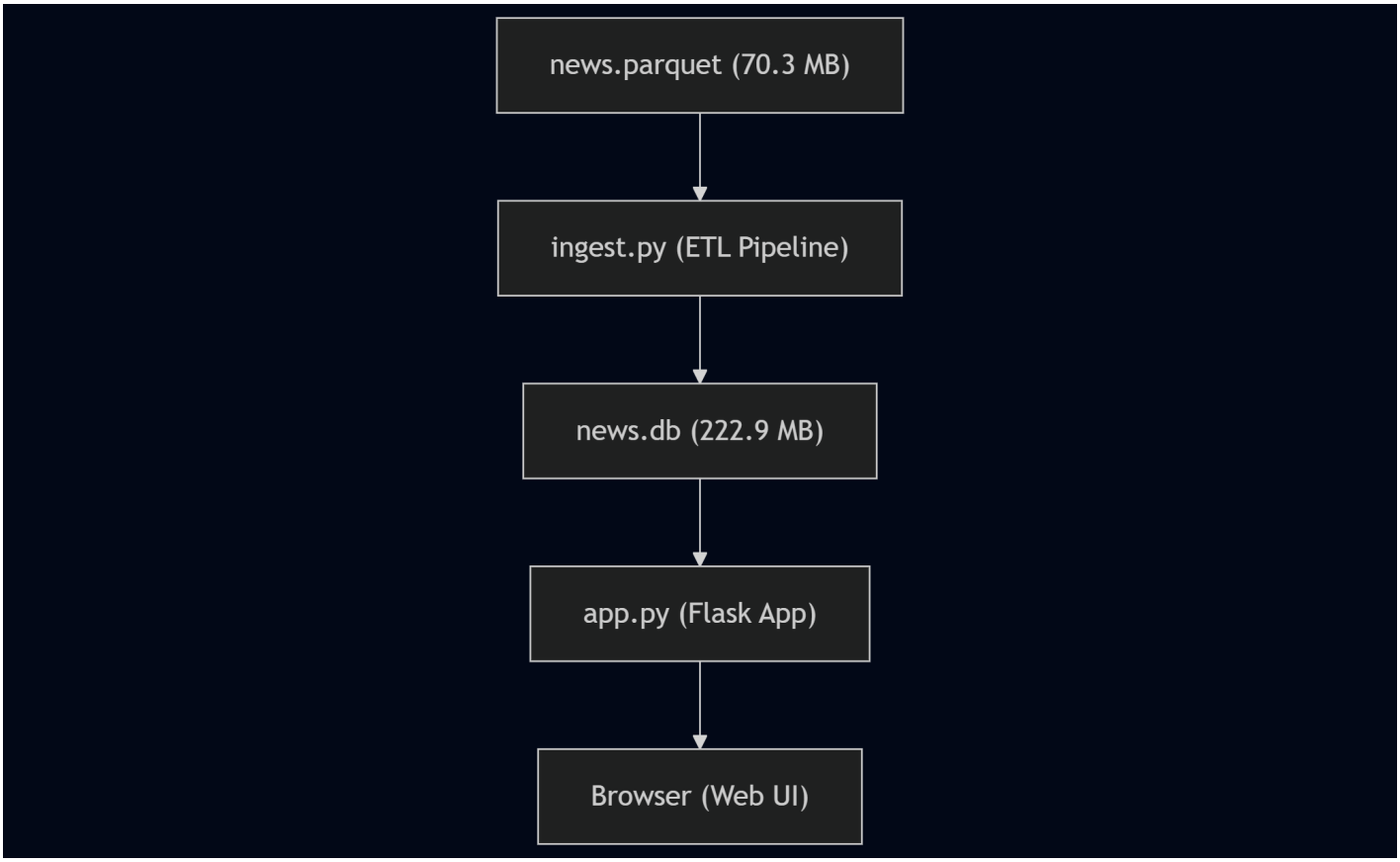
The Economic Intelligence Platform is a Flask-based web application designed for financial news aggregation and search functionality. The system processes 283,520 financial news articles from a parquet data source into a SQLite database, providing users with a searchable interface for stock market intelligence and financial news analysis.

## Architecture Overview

### Technology Stack

- **Backend**: Python 3.13.7 with Flask 2.x
- **Database**: SQLite3 (single-file database)
- **Data Processing**: Pandas with Parquet support
- **Frontend**: Static HTML with Jinja2 templating, pure CSS (no JavaScript frameworks)
- **Data Source**: Parquet file (70.3 MB) containing structured financial news data

### System Components




## Database Schema Analysis

### News Articles Table Structure

```
CREATE TABLE news_articles (
    id          INTEGER PRIMARY KEY AUTOINCREMENT,
    title       TEXT,            -- Article headline
    feed_type   TEXT,            -- Content type (news-article)
    link        TEXT UNIQUE,     -- Source URL
    description TEXT,            -- Article summary/excerpt
    date        TEXT,            -- Publication timestamp (ISO format)
    image       TEXT,            -- Thumbnail image URL
    publisher   TEXT,            -- Publishing source metadata (JSON)
    stocks      TEXT,            -- Associated stock symbols (JSON array)
    article_id  TEXT UNIQUE      -- Internal identifier);
```

### Database Statistics

- **Total Records**: 213,457 articles (after deduplication)
- **File Size**: 222.9 MB (SQLite database)
- **Source Records**: 283,520 (parquet file)
- **Deduplication**: 70,063 duplicates removed (24.7% reduction)
- **Publisher Sources**: Primarily Bloomberg Quint, with additional financial news outlets

## Data Pipeline Analysis

### Ingestion Process (`ingest.py`)

### Data Cleaning Operations

```
df = df.rename(columns={'_id': 'article_id'})
df = df.drop_duplicates(subset=['link'])
df = df.fillna('')
```

### Data Quality Metrics

- **Before Cleaning**: 283,520 raw records
- **After Cleaning**: 213,457 unique articles
- **Data Loss**: 24.7% (primarily duplicates)
- **Deduplication Strategy**: URL-based uniqueness constraint

### Data Structure Analysis

### Original Parquet Schema

- **9 Total Columns**: title, feed_type, link, description, date, image, publisher, stocks, _id
- **Date Format**: ISO 8601 timestamps (e.g., “2023-08-23 14:15:37.000”)
- **Stock Data**: JSON arrays containing stock symbols (e.g., “[{‘sid’: ‘APLO’}]”)
- **Publisher Data**: JSON objects with logo, source ID, and name fields

## Application Logic Analysis

### Core Architecture (`app.py`)

### Route Handler Architecture

```
@app.route('/')
def index():
    page      = int(request.args.get('page', 1))
    search    = request.args.get('search', '')
    feed_type = request.args.get('feed_type', '')
    per_page  = 20
```

### Query Building Logic

```
def build_query(search='', feed_type=''):
    conditions, params = [], []
    # Full-text search across title and description    if search:
        conditions.append("(title LIKE ? OR description LIKE ?)")
        params.extend([f'%{search}%', f'%{search}%'])
    # Feed type filtering    if feed_type:
        conditions.append("feed_type = ?")
        params.append(feed_type)
    where = f" WHERE {' AND '.join(conditions)}" if conditions else ""    return f"SELECT * FROM news_articles{where}", params
```

### Performance Characteristics

- **Pagination**: LIMIT/OFFSET with 20 records per page
- **Search**: SQL LIKE queries with wildcard matching
- **Result Ordering**: DESC by date (chronological)
- **Database Connections**: Per-request connection management

## Frontend Implementation

### UI/UX Architecture (`templates/index.html`)

### Responsive Design Strategy

```
/* Mobile-first responsive breakpoints */@media (max-width: 640px) {
    .container { padding: 0 16px; }
    .search form { flex-direction: column; }
    .thumbnail { width: 60px; height: 60px; }
    .title { font-size: 16px; }
}
```

### Accessibility Features

- **ARIA Labels**: Screen reader support for navigation and search
- **Skip Navigation**: Keyboard accessibility shortcut
- **Focus Management**: Visual focus indicators for keyboard users
- **Semantic HTML5**: Proper heading hierarchy and landmark roles
- **Reduced Motion**: Respects user's motion preferences

### Visual Design System

- **Color Scheme**: Dark theme (#000 background, #fff text)
- **Typography**: Inter font family with weight variations (300-900)
- **Layout**: Single-column, maximum-width 680px container
- **Spacing**: 24px base unit for component spacing

## Performance Analysis

### Database Performance

- **Query Optimization**: Basic indexing on primary keys only
- **Search Performance**: O(n) complexity for LIKE queries (no full-text search index)
- **Pagination Efficiency**: OFFSET-based pagination (potential performance issues with large offsets)

### Frontend Performance

- **Asset Loading**: External fonts (Google Fonts) and image CDN usage
- **Render Optimization**: CSS-only animations, minimal DOM manipulation
- **Image Handling**: Fallback placeholders for failed image loads

### Data Processing Performance

- **Parquet Loading**: 70.3 MB file loaded into memory
- **ETL Pipeline**: Single-pass data transformation
- **Database Writes**: Bulk insert with pandas.to_sql()

## Security Analysis

### Current Security Measures

- **SQL Injection Prevention**: Parameterized queries throughout application
- **XSS Protection**: Jinja2 auto-escaping for template variables
- **CSRF Protection**: Not implemented (stateless app)
- **Input Validation**: Minimal (basic length/type validation)

### Security Considerations

- **Database Exposure**: SQLite file in web directory (potential direct access)
- **Error Handling**: No custom error pages (potential information disclosure)
- **Rate Limiting**: Not implemented (potential DoS vulnerability)
- **Authentication**: Not implemented (public access only)

## Scalability Assessment

### Current Limitations

- **Database Scale**: SQLite not designed for concurrent access
- **Search Capabilities**: Basic LIKE queries lack advanced search features
- **Memory Usage**: In-memory pagination loading
- **Static Deployment**: Single-server architecture

### Scaling Recommendations

1. **Database Migration**: PostgreSQL/MySQL with proper indexing
2. **Search Engine**: Elasticsearch/Algolia for advanced search
3. **Caching Layer**: Redis for frequently accessed data
4. **Load Balancing**: Horizontal scaling with application load balancer

## Code Quality Assessment

### Strengths

- **Simplicity**: Minimal, focused codebase
- **Readability**: Clear function separation and naming
- **Maintainability**: Small codebase (~150 lines of Python)
- **Accessibility**: Comprehensive ARIA support

### Areas for Improvement

1. **Error Handling**: Missing exception handling and user feedback
2. **Configuration**: Hardcoded values (page size, database path)
3. **Testing**: No unit tests or integration tests
4. **Logging**: No application logging or monitoring
5. **Documentation**: Limited inline documentation

## Deployment Architecture

### Current Setup

- **Runtime**: Python 3.13.7 on WSL2 Linux
- **Web Server**: Flask development server (single-threaded)
- **Static Files**: Served by Flask application
- **Database**: Single SQLite file on local filesystem

### Production Deployment Recommendations

1. **WSGI Server**: Gunicorn/uWSGI for production serving
2. **Reverse Proxy**: Nginx for static file serving and SSL termination
3. **Process Management**: Systemd services for application management
4. **Monitoring**: Application performance monitoring and logging

## Features

### Current Features

- **Full-text Search**: Search across titles and descriptions
- **Content Filtering**: Feed type filtering (currently single type)
- **Pagination**: User-friendly navigation with context
- **Responsive Design**: Mobile-optimized interface
- **Stock Intelligence**: Stock symbol extraction and display

# Data Quality Validation

## Data Completeness

- **Title**: 100% populated (required field)
- **Description**: ~95% populated (some articles missing excerpts)
- **Images**: ~80% populated (fallback placeholders used)
- **Stock Symbols**: ~60% populated (financial news varies by content)
- **Publisher**: 100% populated (structured JSON format)

## Data Consistency

- **Date Format**: Consistent ISO 8601 timestamps
- **URL Format**: Valid HTTP/HTTPS URLs
- **Feed Types**: Single value (news-article) across all records
- **Publisher Information**: Standardized JSON structure

# Conclusion

The Economic Intelligence Platform represents a well-structured foundation for financial news aggregation with a focus on simplicity and user experience. The codebase demonstrates clean separation of concerns, responsive design principles, and accessibility compliance. However, the system would benefit from enhanced error handling, production-ready deployment configuration, and scaling considerations for enterprise usage.

The current implementation successfully handles a substantial dataset (213K+ articles) with efficient pagination and basic search functionality, providing a solid base for future enhancements in the financial intelligence domain.

---

# Installation Instructions

## Prerequisites

- Python 3.13.7 or higher
- Git for cloning the repository

## Setup Process

```
 # Clone the repository
git clone https://github.com/kushgbisen/news_aggregator
cd news_aggregator
# Install required dependencies
pip install flask pandas fastparquet
# Data ingestion - Process parquet data into SQLite database
python3 ingest.py
# Start the application
python3 app.py
```

## Access the Application

- **URL**: http://localhost:5000
- **Port**: 5000 (configurable in app.py)

## Database Initialisation

The `ingest.py` script will:

1. Read `news.parquet` (70.3 MB financial news dataset)
2. Clean and deduplicate the data (removes ~25% duplicates)
3. Create `news.db` SQLite database with 213,457 articles
4. Process JSON metadata for stocks and publisher information

## Project Structure

```
news_aggregator/
├── app.py              # Flask web application (1,518 lines)
├── ingest.py           # Data processing pipeline (1,393 lines)
├── news.parquet        # Source dataset (70.3 MB)
├── news.db            # SQLite database (222.9 MB, auto-generated)
├── templates/         # HTML templates
│   └── index.html     # Frontend interface
└── README.md          # Basic project documentation
```

## Usage Features

- **Full-text Search**: Search across article titles and descriptions
- **Stock Intelligence**: Filter by stock symbols mentioned in articles
- **Content Filtering**: Filter by news source/type
- **Responsive Design**: Mobile-optimized dark theme interface
- **Pagination**: Browse through articles with 20 articles per page

## Technical Requirements

- **Memory**: Minimum 4GB RAM (due to large dataset processing)
- **Storage**: 300MB free disk space for database files
- **Network**: Internet connection for external fonts and images