# Children's News App Blueprint

**Data Extraction & Processing Blueprint**

**Project:** Data Pipeline for Child-Friendly News  
**Author:** [Your Name]  
**Date:** October 2025  
**Duration:** 24 man-hours

**1. Overview of Data Sources**

**1.1 Source Selection Strategy**

News sources were evaluated based on accessibility (RSS/API availability), content quality, update frequency, and technical feasibility. From 10 identified sources, 2 were implemented for the PoC.

**1.2 Identified Sources (8-10 Total)**

**National (India):**

1. The Hindu - RSS available
2. Hindustan Times - Scrapable
3. Indian Express - RSS available
4. NDTV - RSS available
5. Times of India - RSS available
6. The Wire - Scrapable

**International:** 7. BBC News Asia - RSS available 8. Al Jazeera - RSS available 9. The Guardian - RSS available 10. Reuters - API/RSS (premium)

**1.3 Implemented Sources**

| **Source** | **Method** | **URL** | **Articles/Run** | **Rationale** |
| --- | --- | --- | --- | --- |
| **The Hindu** | RSS Feed | thehindu.com/rss | 10 | Reliable XML structure, quality journalism, full content extraction |
| **Hindustan Times** | Web Scraping | hindustantimes.com | 10 | Wide coverage, clean HTML, no anti-scraping measures |

**Why This Combination?**

* Demonstrates both RSS and scraping techniques
* RSS provides reliability; scraping shows versatility
* Combined output: 20 articles per execution

**Challenges Encountered:**

* Reuters: 401 Forbidden errors due to bot detection
* BBC India: Dynamic JavaScript content loading
* Solution: Selected stable, scrapable alternatives

**2. System Architecture & Data Flow**

**2.1 High-Level Architecture**

┌─────────────────────────────────────────────────────┐

│ DATA SOURCES (2) │

│ The Hindu (RSS) | Hindustan Times (Scraper) │

└───────────────┬─────────────────────────────────────┘

│

▼

┌─────────────────────────────────────────────────────┐

│ EXTRACTION LAYER (extractors.py) │

│ • RSSExtractor: Parse RSS feeds │

│ • WebScraper: Scrape HTML pages │

│ • Output: raw\_articles.json (20 articles) │

└───────────────┬─────────────────────────────────────┘

│

▼

┌─────────────────────────────────────────────────────┐

│ PROCESSING LAYER (processors.py) │

│ Step 1: Data Cleaning │

│ • Text normalization │

│ • Deduplication (title-based) │

│ • Sensitive content filtering │

│ Output: cleaned\_articles.json (6-12 articles) │

│ │

│ Step 2: Mock AI Processing │

│ • Word simplification (complex → simple) │

│ • Sentence shortening │

│ • Generate 3 age-group versions │

│ Output: processed\_articles.json (18-36 versions) │

└───────────────┬─────────────────────────────────────┘

│

▼

┌─────────────────────────────────────────────────────┐

│ STORAGE LAYER (database.py) │

│ SQLite Database (news\_database.db) │

│ • raw\_articles (20 records) │

│ • cleaned\_articles (6-12 records) │

│ • processed\_articles (18-36 records) │

│ • extraction\_stats (metadata) │

└─────────────────────────────────────────────────────┘

**2.2 Pipeline Execution Flow**

**Complete Pipeline (data\_pipeline.py):**

1. **Extraction (2-3 min):** Fetch RSS XML → Parse metadata → Visit article URLs → Extract full text → Save JSON
2. **Cleaning (10-20 sec):** Load raw data → Normalize text → Remove duplicates → Filter sensitive content → Save JSON
3. **Processing (20-30 sec):** Load cleaned data → Simplify words → Shorten sentences → Generate age versions → Save JSON
4. **Storage (5-10 sec):** Connect DB → Insert records → Update statistics

**Total Execution Time:** ~3-4 minutes for 20 articles

**3. Database Schema & Structure**

**3.1 Schema Design (SQLite)**

**4 Tables with Referential Integrity:**

**Table 1: raw\_articles**

CREATE TABLE raw\_articles (

id INTEGER PRIMARY KEY AUTOINCREMENT,

source TEXT NOT NULL,

title TEXT NOT NULL,

url TEXT UNIQUE,

raw\_content TEXT,

published\_date TEXT,

extraction\_method TEXT,

created\_at TEXT DEFAULT CURRENT\_TIMESTAMP

);

**Purpose:** Store unprocessed articles as extracted

**Table 2: cleaned\_articles**

CREATE TABLE cleaned\_articles (

id INTEGER PRIMARY KEY AUTOINCREMENT,

raw\_article\_id INTEGER,

title TEXT NOT NULL,

raw\_content TEXT,

word\_count INTEGER,

cleaned\_at TEXT,

FOREIGN KEY (raw\_article\_id) REFERENCES raw\_articles(id)

);

**Purpose:** Store safe, filtered articles

**Table 3: processed\_articles**

CREATE TABLE processed\_articles (

id INTEGER PRIMARY KEY AUTOINCREMENT,

cleaned\_article\_id INTEGER,

title TEXT NOT NULL,

age\_group\_key TEXT NOT NULL,

age\_group\_name TEXT,

simplified\_text TEXT,

word\_count INTEGER,

sentence\_count INTEGER,

FOREIGN KEY (cleaned\_article\_id) REFERENCES cleaned\_articles(id)

);

**Purpose:** Store age-specific versions (3 per article)

**Table 4: extraction\_stats**

CREATE TABLE extraction\_stats (

id INTEGER PRIMARY KEY AUTOINCREMENT,

extraction\_date TEXT,

total\_raw\_articles INTEGER,

total\_cleaned\_articles INTEGER,

sources\_used TEXT

);

**Purpose:** Track pipeline execution metrics

**3.2 Entity Relationships**

raw\_articles (1) ──→ (N) cleaned\_articles (1) ──→ (3) processed\_articles

**Example Data Flow:**

* 1 raw article → 1 cleaned article → 3 processed versions (one per age group)
* 20 raw articles → 12 cleaned (8 filtered) → 36 processed (12 × 3)

**3.3 Sample Query**

-- Get all simplified articles for 6-8 year olds

SELECT title, simplified\_text, word\_count

FROM processed\_articles

WHERE age\_group\_key = 'group\_1'

ORDER BY created\_at DESC;

**4. AI Model Input/Output Design**

**4.1 Mock AI Approach**

For this PoC, a **deterministic rule-based processor** simulates AI functionality. This demonstrates the pipeline without requiring ML model training.

**4.2 Age Group Configuration**

| **Age Group** | **Max Words** | **Max Sentences** | **Complexity** |
| --- | --- | --- | --- |
| 6-8 years | 50 | 3 | Very Simple |
| 9-11 years | 100 | 5 | Simple |
| 12-14 years | 150 | 8 | Moderate |

**4.3 Processing Steps**

**Step 1: Word Simplification**

* Dictionary-based replacement (30+ mappings)
* Examples: "approximately" → "about", "consequently" → "so", "demonstrate" → "show"

**Step 2: Sentence Shortening**

* Split long sentences at commas, conjunctions
* Break sentences >15 words into smaller chunks

**Step 3: Length Limiting**

* Enforce max words per age group
* Preserve first N most important sentences
* Add "..." if truncated

**4.4 Sensitive Content Filtering**

**Keyword-based filtering** removes articles containing:

murder, killed, death, violence, terrorist, bomb, rape,

assault, abuse, suicide, war, shooting, attack, etc.

**Effectiveness:** Filters 30-70% of articles depending on news cycle

**4.5 Input/Output Example**

**Input (Raw Article - 185 words):**

"The Karnataka government's ambitious tunnel road project in Bengaluru has encountered significant criticism from an expert committee. The committee concluded that the DPR had been prepared hastily, relying on merely four soil test points..."

**Output for 6-8 years (45 words):**

"The Karnataka government has a big tunnel road plan in Bengaluru. A group of experts said the plan was made too fast. They said more tests are needed..."

**Output for 9-11 years (98 words):**

"The Karnataka government wants to build a tunnel road in Bengaluru. But a group of experts found problems with the plan. The experts said the plan was made in a hurry. They said the builders only did four soil tests. That's not enough for such a big project..."

**Output for 12-14 years (148 words):**

"The Karnataka government's ambitious tunnel road project in Bengaluru has faced criticism from an expert committee. The committee reviewed the plan and found several problems. First, the plan was prepared hastily with only four soil test points—far too few for a project of this scale..."

**4.6 Production AI Path**

**Future Enhancement:** Replace mock processor with:

* Fine-tuned BERT/GPT models for text simplification
* Extractive/abstractive summarization
* Readability scoring (Flesch-Kincaid)
* Named Entity Recognition to preserve key facts

**5. Key Limitations & Risks**

**5.1 Technical Limitations**

**Data Extraction:**

* RSS feeds limited to source update frequency (15 min - 1 hour)
* Web scraping breaks when sites redesign (requires maintenance)
* Rate limiting possible with multiple sources (mitigated with delays)

**Mock AI:**

* Not true AI; simple rule-based logic
* May oversimplify or lose context
* Grammar issues from sentence splitting
* Limited word simplification dictionary

**Sensitive Filtering:**

* Keyword-based; may miss euphemisms or context
* Over-filtering (30-70%) reduces available content
* Only works for English

**Database:**

* SQLite not ideal for concurrent writes (production needs PostgreSQL)
* Single-file architecture limits scalability

**5.2 Operational Risks**

**Source Reliability:**

* Website downtime stops extraction (mitigated with try-except)
* Anti-scraping measures may block requests (encountered with Reuters)

**Content Appropriateness:**

* Keyword filter may miss implicit sensitive content
* Risk: Exposing children to inappropriate material
* Mitigation: Conservative filtering, manual review recommended

**Data Quality:**

* Duplicate content from multiple sources (current: title-based deduplication)
* No fact-checking in pipeline (relies on source credibility)

**5.3 Challenges Faced**

1. **Full Content Extraction:** Initial RSS feeds only provided summaries (50-100 words). Solution: Implemented secondary URL fetching to scrape full article text.
2. **Web Scraping Stability:** Reuters returned 401 errors, BBC had dynamic content. Solution: Switched to Hindustan Times with stable HTML.
3. **Over-filtering:** Initial keyword list filtered 80%+ articles. Solution: Refined to focus on explicit violence/abuse.

**6. Tools, Libraries & Environment**

**6.1 Technology Stack**

**Language:** Python 3.8+  
**Database:** SQLite (migration path to PostgreSQL)  
**Development:** Visual Studio Code, Git

**6.2 Core Libraries**

| **Library** | **Version** | **Purpose** |
| --- | --- | --- |
| feedparser | 6.0.10 | RSS feed parsing |
| beautifulsoup4 | 4.12.2 | HTML parsing for scraping |
| requests | 2.31.0 | HTTP requests |
| lxml | 4.9.3 | Fast XML/HTML parser |
| sqlalchemy | 2.0.23 | Database ORM (optional) |
| pandas | 2.1.3 | Data manipulation (optional) |

**Why These Tools?**

* **feedparser:** Industry standard, handles malformed feeds
* **BeautifulSoup:** Easy to learn, excellent for web scraping
* **SQLite:** Zero-config database, perfect for PoC

**6.3 Project Structure**

children\_news\_app/

│

├── data/ # JSON outputs

│ ├── raw\_articles.json

│ ├── cleaned\_articles.json

│ └── processed\_articles.json

│

├── src/ # Source code

│ ├── config.py # Configuration

│ ├── extractors.py # Data extraction

│ ├── processors.py # Cleaning & AI

│ ├── database.py # Database ops

│ └── data\_pipeline.py # Orchestrator

│

├── news\_database.db # SQLite database

├── requirements.txt # Dependencies

└── README.md # Setup guide

**6.4 Setup & Installation**

# 1. Create virtual environment

python -m venv venv

source venv/bin/activate # Windows: venv\Scripts\activate

# 2. Install dependencies

pip install -r requirements.txt

# 3. Run complete pipeline

python src/data\_pipeline.py full

**Execution Time:** 3-4 minutes for complete pipeline

**7. Scaling & Future Improvements**

**7.1 Immediate Enhancements (1-3 months)**

**Add More Sources:**

* Expand to 10-15 sources (NDTV, Indian Express, BBC, etc.)
* Multi-language support (Hindi, Tamil)
* Estimated effort: 2-3 days per source

**Improve Filtering:**

* Machine learning classifier for appropriate/inappropriate content
* Topic categorization (Science, Sports, Culture vs. Politics, Crime)
* Estimated effort: 1-2 weeks

**Better AI Processing:**

* Implement readability scoring (Flesch-Kincaid Grade Level)
* Extractive summarization (TextRank, LSA)
* Named Entity Recognition to preserve key facts
* Estimated effort: 1 week

**7.2 Production Features (3-6 months)**

**Database Migration:**

* PostgreSQL for better concurrency (1000s simultaneous users)
* Connection pooling, replication
* Estimated effort: 3-5 days

**API Development:**

* RESTful API with FastAPI
* Endpoints: GET /articles, GET /articles/{id}, filter by age/category
* JWT authentication, rate limiting
* Estimated effort: 1 week

**Real-Time Processing:**

* Event-driven architecture (RabbitMQ/Kafka)
* Process articles as published, not batch
* Horizontal scaling with worker pools
* Estimated effort: 2 weeks

**7.3 Advanced Features (6-12 months)**

**Production AI Models:**

* Fine-tuned BERT for text simplification
* Training dataset: 10,000-50,000 article pairs
* Expected BLEU score: 80-85%
* Estimated effort: 2-3 months

**Multimedia Content:**

* Image extraction and filtering
* Video summarization
* Text-to-speech audio articles
* Estimated effort: 3-4 months

**Personalization:**

* User profiles (age, interests, reading level)
* Recommendation engine
* Adaptive complexity
* Estimated effort: 2-3 months

**7.4 Scalability Roadmap**

| **Phase** | **Timeline** | **Sources** | **Articles/Day** | **Users** | **Infrastructure Cost** |
| --- | --- | --- | --- | --- | --- |
| **PoC** (Current) | Complete | 2 | 20 | 0 | ₹0 |
| **Small Scale** | 1-3 months | 10 | 200+ | <1,000 | ₹4,000/month |
| **Medium Scale** | 3-6 months | 20-30 | 1,000+ | 1K-10K | ₹17,000/month |
| **Large Scale** | 6-12 months | 50+ | 5,000+ | 10K-100K | ₹60,000/month |

**7.5 Technology Migration**

| **Component** | **Current (PoC)** | **Production** |
| --- | --- | --- |
| Database | SQLite | PostgreSQL |
| Caching | None | Redis |
| Message Queue | None | RabbitMQ |
| Deployment | Manual | Docker + Kubernetes |
| Monitoring | Prints | Prometheus + Grafana |

**8. Conclusion**

This PoC successfully demonstrates an end-to-end data pipeline for child-friendly news content. Key achievements include:

✅ Dual extraction methods (RSS + scraping)  
✅ Intelligent content filtering (30-70% sensitive articles removed)  
✅ Multi-age-group processing (3 versions per article)  
✅ Structured database storage  
✅ Fully automated execution

**Production Readiness:** The modular architecture enables easy enhancement. Priority next steps: expand sources, implement ML models, and deploy REST API for mobile app integration.

**Business Potential:** Growing demand for age-appropriate content, educational partnerships, subscription model viability.