**Cursor AI Prompts for Mingus Article Library Implementation**

**📧 PHASE 1: Email & Article Extraction**

**Prompt 1: Email Article URL Extractor**

Create a Python script called `scripts/extract\_article\_urls.py` for the Mingus financial wellness app that:

CONTEXT: Mingus is a Flask/PostgreSQL app serving African American professionals aged 25-35. I have emails containing financial article links that need to be extracted and organized.

REQUIREMENTS:

1. Extract all URLs from email text/HTML content

2. Filter for reputable financial websites (nerdwallet.com, bankrate.com, thebalance.com, investopedia.com, blackenterprise.com, essence.com, theroot.com, mint.com, kiplinger.com, forbes.com, harvard.edu, medium.com, linkedin.com/pulse)

3. Validate URLs are still accessible (200 status check)

4. Handle different email formats (.mbox, .eml, or raw text)

5. Output to CSV with columns: url, email\_subject, email\_date, sender, domain

6. Include progress tracking and error handling

7. Remove duplicates and sort by domain

ADDITIONAL FEATURES:

- Rate limiting for URL validation (1 request per second)

- Detailed logging of extraction process

- Option to process single email file or batch directory

- Command line interface for easy execution

Generate clean, documented Python code with proper error handling.

**Prompt 2: Article Content Scraper**

Create a Python script called `scripts/scrape\_article\_content.py` for Mingus that:

CONTEXT: Mingus financial app needs to scrape full article content from validated URLs for AI classification. Target audience is African American professionals earning $40K-$100K.

REQUIREMENTS:

1. Use newspaper3k and beautifulsoup4 for content extraction

2. Extract: title, full text content, author, publish\_date, meta\_description, top\_image, word\_count

3. Handle different website structures and content formats

4. Implement robust error handling for failed scrapes

5. Rate limiting (2 seconds between requests) to respect websites

6. Input: CSV file with article URLs

7. Output: Enhanced CSV with all extracted content

8. Progress tracking with percentage completion

9. Retry logic for failed extractions (3 attempts)

10. Filter out articles with <300 or >5000 words

SPECIAL HANDLING:

- Detect and handle paywalled content gracefully

- Extract clean text without ads/navigation

- Preserve article structure and formatting

- Log successful vs failed extractions

- Create backup JSON file of all extracted data

Include comprehensive logging and recovery mechanisms for interrupted processing.

**🤖 PHASE 2: AI Classification System**

**Prompt 3: OpenAI Article Classifier**

Create `scripts/ai\_article\_classifier.py` for Mingus that uses OpenAI GPT-4 to classify articles:

CONTEXT: Mingus uses a "Be-Do-Have" framework where:

- BE = Identity transformation, mindset, confidence, professional identity for African American professionals

- DO = Skill development, actions, networking, career advancement, negotiation

- HAVE = Income achievement, wealth building, financial management, lifestyle optimization

TARGET USERS: African American professionals, 25-35 years old, $40K-$100K income, facing challenges like student debt, career barriers, homeownership obstacles.

REQUIREMENTS:

1. Create detailed OpenAI prompt that understands the cultural context

2. Classify each article into primary phase (BE/DO/HAVE)

3. Assign gate level (Beginner/Intermediate/Advanced) based on prerequisite knowledge

4. Rate demographic relevance (1-10) for target audience

5. Extract 3-5 key topics covered

6. Identify prerequisites users should have

7. Suggest recommended order in sequence (1-100)

8. Provide reasoning for classification decisions

API FEATURES:

- Use GPT-4 for accuracy, fallback to GPT-3.5-turbo for cost savings

- Batch processing with progress tracking

- JSON output format for database integration

- Error handling and retry logic

- Cost estimation and token usage tracking

- Save intermediate results to prevent data loss

OUTPUT FORMAT: Structured JSON with all classification data

INPUT: CSV file with scraped article content

RATE LIMITING: 1 request per second to avoid API limits

Generate robust code with comprehensive error handling and cost optimization.

**Prompt 4: Classification Validation System**

Create `scripts/validate\_classifications.py` for Mingus that:

CONTEXT: After AI classification, need human validation system to ensure accuracy for African American professional audience.

REQUIREMENTS:

1. Load classified articles from CSV/JSON

2. Present random sample for manual review (suggest 20% of total)

3. Show article title, content preview, and AI classification

4. Allow human reviewer to accept, modify, or reject AI classification

5. Track validation statistics (accuracy rate, common errors)

6. Update classifications based on human feedback

7. Generate validation report with recommendations

8. Export corrected classifications back to CSV

UI FEATURES:

- Command-line interface with clear article display

- Easy accept/reject/modify workflow

- Progress tracking through validation queue

- Ability to save progress and resume later

- Quick keyboard shortcuts for common actions

- Display AI confidence scores when available

ANALYTICS:

- Calculate AI accuracy by category (BE/DO/HAVE)

- Identify patterns in misclassifications

- Export validation metrics for process improvement

- Generate feedback for improving AI prompts

Make the validation process efficient and user-friendly for processing 100+ articles.

**🗄️ PHASE 3: Database Integration**

**Prompt 5: Database Schema Migration**

Create `migrations/add\_article\_library.py` for Mingus Flask app that:

CONTEXT: Mingus uses PostgreSQL with SQLAlchemy. Existing tables include users, user\_profiles. Need to add article library with Be-Do-Have framework support.

REQUIREMENTS:

1. Create articles table with UUID primary keys (consistent with existing schema)

2. Support Be-Do-Have classification (BE/DO/HAVE enum)

3. Gate levels (Beginner/Intermediate/Advanced enum)

4. Full-text search using PostgreSQL tsvector

5. User progress tracking for articles

6. Assessment scores for gatekeeping

7. Bookmark/favorite functionality

8. Reading analytics (time spent, completion rate)

TABLES NEEDED:

- articles (content, classification, metadata)

- user\_article\_progress (reading progress, bookmarks, ratings)

- user\_assessment\_scores (Be-Do-Have scores for gatekeeping)

- article\_categories (for future organization)

INDEXES:

- Full-text search on article content

- Performance indexes on classification fields

- User progress lookup optimization

CONSTRAINTS:

- Proper foreign key relationships

- Data validation constraints

- Unique constraints where appropriate

Generate Alembic-compatible migration with rollback capability. Include sample data insertion for testing.

**Prompt 6: SQLAlchemy Models**

Create SQLAlchemy models in `backend/models/articles.py` for Mingus that:

CONTEXT: Mingus Flask app with existing User and UserProfile models. Need article library models that integrate with Be-Do-Have framework.

REQUIREMENTS:

1. Article model with all fields from migration

2. UserArticleProgress model for tracking user interactions

3. UserAssessmentScores model for gatekeeping logic

4. Proper relationships between models

5. Class methods for common queries (search, recommendations)

6. Properties for calculated fields (reading\_completion\_rate, etc.)

7. Serialization methods for API responses

KEY FEATURES:

- Full-text search methods using PostgreSQL

- Gatekeeping logic (can\_user\_access\_article)

- Progress tracking methods

- Recommendation algorithms based on user profile

- Analytics methods for admin dashboard

METHODS TO INCLUDE:

- Article.search(query, filters, user\_id)

- Article.get\_recommendations(user\_id, limit)

- UserArticleProgress.mark\_completed()

- UserAssessmentScores.calculate\_readiness\_level()

- Article.get\_by\_phase\_and\_level()

Integrate with existing Mingus models and maintain consistent coding patterns. Include proper docstrings and type hints.

**🔌 PHASE 4: Backend API Development**

**Prompt 7: Article Management API Routes**

Create `backend/routes/articles.py` for Mingus Flask app that:

CONTEXT: Mingus serves African American professionals with Be-Do-Have personal transformation framework. Users have assessment scores that gate access to articles.

API ENDPOINTS NEEDED:

1. GET /api/articles - List articles with filtering (phase, level, search)

2. GET /api/articles/<id> - Get single article with progress tracking

3. POST /api/articles/<id>/progress - Update reading progress

4. POST /api/articles/<id>/bookmark - Toggle bookmark status

5. GET /api/articles/recommendations - Personalized recommendations

6. GET /api/articles/search - Advanced search with filters

7. GET /api/user/assessment - Get user's Be-Do-Have scores

8. POST /api/user/assessment - Update assessment scores

FEATURES:

- Authentication required for all endpoints

- Gatekeeping logic (check user assessment vs article requirements)

- Progress tracking integration

- Search with PostgreSQL full-text search

- Pagination for large result sets

- Error handling with appropriate HTTP status codes

- Rate limiting for search endpoints

RESPONSE FORMATS:

- Consistent JSON structure

- Include user progress data where applicable

- Metadata for pagination

- Access control flags (can\_access, is\_bookmarked, etc.)

Use Flask-RESTful patterns and integrate with existing Mingus authentication system.

**Prompt 8: Article Search Service**

Create `backend/services/article\_search.py` for Mingus that provides advanced search capabilities:

CONTEXT: Mingus users need to find relevant articles based on their Be-Do-Have progress, financial goals, and personal situation.

SEARCH FEATURES:

1. Full-text search across title, content, and topics

2. Filter by Be-Do-Have phase

3. Filter by gate level (based on user assessment)

4. Filter by demographic relevance score

5. Filter by reading time/word count

6. Personalized ranking based on user profile

7. Trending articles (based on recent user activity)

8. Similar articles recommendation

RANKING ALGORITHM:

- Text relevance score (PostgreSQL ts\_rank)

- Demographic match score for target audience

- User personalization (based on goals, industry, income level)

- Article quality score

- Recency boost for newer content

PERFORMANCE:

- Efficient database queries with proper indexing

- Caching for common searches using Redis

- Pagination for large result sets

- Search analytics for improving relevance

INTEGRATION:

- Works with existing user profile data

- Respects gatekeeping rules

- Tracks search analytics for admin insights

Create a robust search service that helps users discover the most relevant transformation content.

**🎨 PHASE 5: Frontend React Components**

**Prompt 9: Article Library Main Component**

Create `frontend/components/ArticleLibrary.jsx` for Mingus React frontend that:

CONTEXT: Mingus uses React with Tailwind CSS. Target users are African American professionals using Be-Do-Have transformation framework. App has existing authentication and user profile system.

COMPONENT REQUIREMENTS:

1. Phase navigation (BE/DO/HAVE tabs) with progress indicators

2. Article grid with cards showing title, description, topics, and access status

3. Search functionality with filters (phase, level, topics)

4. Gatekeeping UI (locked articles with upgrade prompts)

5. Progress tracking (reading completion, bookmarks)

6. Responsive design for mobile and desktop

7. Loading states and error handling

UI FEATURES:

- Clean, professional design matching Mingus brand

- Visual indicators for locked/unlocked content

- Progress bars for reading completion

- Bookmark/favorite functionality

- Article difficulty indicators (Beginner/Intermediate/Advanced)

- Cultural relevance scores displayed

- Quick action buttons (Read, Bookmark, Share)

STATE MANAGEMENT:

- React hooks for component state

- API integration for fetching articles

- User assessment scores for gatekeeping

- Search and filter state management

- Progress tracking updates

ACCESSIBILITY:

- ARIA labels and semantic HTML

- Keyboard navigation support

- Screen reader compatibility

- Color contrast compliance

Generate modern React component with Tailwind CSS styling that feels native to the Mingus app experience.

**Prompt 10: Assessment Integration Component**

Create `frontend/components/BeDoHaveAssessment.jsx` for Mingus that:

CONTEXT: Need quick assessment to determine user readiness for Be-Do-Have article phases. Integrates with existing weekly check-in system.

ASSESSMENT DESIGN:

1. 15 questions total (5 per phase: BE/DO/HAVE)

2. 5-point Likert scale responses

3. Progress indicator and completion tracking

4. Automatic scoring and level assignment

5. Results display with unlock notifications

6. Integration with article gatekeeping system

BE QUESTIONS (Identity/Mindset):

- Professional confidence and executive presence

- Overcoming limiting beliefs and imposter syndrome

- Career vision and goal clarity

- Leadership mindset development

- Cultural identity integration in professional settings

DO QUESTIONS (Skills/Actions):

- Skill development activities

- Networking and relationship building

- Career advancement actions

- Income generation strategies

- Professional development investments

HAVE QUESTIONS (Results/Management):

- Current income and savings levels

- Wealth building activities

- Financial management skills

- Investment knowledge and experience

- Money mindset and lifestyle management

UI FEATURES:

- Clean, engaging question flow

- Visual progress tracking

- Immediate score calculation

- Results visualization (radar chart or progress bars)

- Article unlock notifications

- Retake capability for score improvement

Generate assessment component that feels motivational and educational, not just evaluative.

**Prompt 11: Article Card Component**

Create `frontend/components/ArticleCard.jsx` for Mingus that displays individual articles:

CONTEXT: Mingus targets African American professionals with culturally-relevant financial content. Cards need to show value and accessibility clearly.

CARD FEATURES:

1. Article title and description

2. Be-Do-Have phase badge with distinct colors

3. Difficulty level indicator (Beginner/Intermediate/Advanced)

4. Demographic relevance score (styled as "match percentage")

5. Key topics as tags

6. Reading time estimate

7. Access status (locked/unlocked with clear messaging)

8. Bookmark/favorite toggle

9. Progress indicator if started

10. Social sharing options

VISUAL DESIGN:

- Card layout with clean typography

- Color coding for Be-Do-Have phases (e.g., BE=blue, DO=green, HAVE=gold)

- Lock/unlock icons with clear status

- Progress bars for reading completion

- Bookmark heart icon with animation

- Hover effects and interaction feedback

FUNCTIONALITY:

- Click to read (if unlocked) or show unlock requirements

- Bookmark toggle with API integration

- Progress tracking when user returns

- Share functionality for social media

- "Why this matters" tooltip for locked content

ACCESSIBILITY:

- Semantic HTML structure

- ARIA labels for interactive elements

- Keyboard navigation support

- Screen reader friendly descriptions

STATES TO HANDLE:

- Locked (show requirements to unlock)

- Unlocked but not started

- In progress (show completion percentage)

- Completed (show completion badge)

- Bookmarked status

Create an engaging card component that motivates users to engage with the transformation content.

**📊 PHASE 6: Data Import & Testing**

**Prompt 12: Article Data Import Script**

Create `scripts/import\_articles\_to\_database.py` for Mingus that:

CONTEXT: Need to import classified articles from CSV into PostgreSQL database with proper data validation and error handling.

REQUIREMENTS:

1. Read classified articles from CSV file

2. Validate all required fields are present

3. Clean and normalize data (trim whitespace, validate URLs, etc.)

4. Handle duplicate detection and resolution

5. Import to articles table using SQLAlchemy models

6. Create initial assessment scores for existing users

7. Generate import report with statistics

8. Handle errors gracefully with detailed logging

DATA VALIDATION:

- URL format validation and accessibility check

- Title and content length validation

- Phase values must be BE/DO/HAVE

- Gate level must be Beginner/Intermediate/Advanced

- Demographic match score must be 1-10

- JSON validation for topics and prerequisites

ERROR HANDLING:

- Skip invalid records with detailed error logging

- Continue processing on individual failures

- Generate report of skipped records for manual review

- Rollback capability if critical errors occur

FEATURES:

- Progress tracking with percentage completion

- Batch processing for large datasets

- Dry-run mode to validate before actual import

- Backup existing data before import

- Command-line interface with options

POST-IMPORT TASKS:

- Update search vectors for full-text search

- Generate sample user assessment scores

- Create initial article categories

- Verify foreign key relationships

Generate robust import script that can handle 100-500 articles reliably.

**Prompt 13: End-to-End Testing Suite**

Create comprehensive test suite in `tests/test\_article\_library.py` for Mingus that:

CONTEXT: Testing article library feature for African American professional users with Be-Do-Have framework and assessment gatekeeping.

TEST CATEGORIES:

1. Article model tests (creation, validation, relationships)

2. User assessment scoring tests

3. Gatekeeping logic tests (can user access article)

4. Search functionality tests

5. API endpoint tests (all CRUD operations)

6. Frontend component tests (rendering, interactions)

7. Integration tests (full user journey)

KEY TEST SCENARIOS:

- User completes assessment and unlocks articles

- Article search with various filters and queries

- Reading progress tracking and completion

- Bookmark functionality across sessions

- Recommendation engine accuracy

- Database constraints and data validation

- Authentication and authorization

- Performance with large datasets

EDGE CASES:

- Users with no assessment scores

- Articles with missing or invalid data

- Search queries with special characters

- Network failures during API calls

- Concurrent user access to same articles

MOCKING:

- Mock OpenAI API calls for classification

- Mock external article URLs for scraping

- Mock user authentication for testing

- Mock database for isolated unit tests

TEST DATA:

- Sample articles for each Be-Do-Have phase

- User profiles with different assessment scores

- Search queries representing real user behavior

- Edge case data for validation testing

Generate comprehensive test suite using pytest with proper fixtures, mocking, and assertion patterns. Include performance benchmarks for search and recommendation features.

**🚀 PHASE 7: Analytics & Monitoring**

**Prompt 14: Article Analytics Service**

Create `backend/services/article\_analytics.py` for Mingus that tracks and analyzes article engagement:

CONTEXT: Need comprehensive analytics for article library to optimize content and user experience for African American professionals.

ANALYTICS FEATURES:

1. Article performance metrics (views, completion rate, time spent)

2. User engagement patterns (reading habits, preferred phases)

3. Search analytics (popular queries, success rates)

4. Assessment score progression tracking

5. Content gap analysis (unmet user needs)

6. Recommendation effectiveness measurement

METRICS TO TRACK:

- Article views and unique readers

- Reading completion rates by article and category

- Average reading time and engagement depth

- Bookmark and sharing rates

- Search query analysis and success rates

- User progression through Be-Do-Have phases

- Assessment score improvements over time

- Revenue impact (correlation with subscription retention)

REPORTING FEATURES:

- Daily/weekly/monthly analytics dashboards

- Content performance rankings

- User segment analysis (by income, goals, demographics)

- Trend analysis for content optimization

- Export capabilities for external analysis

INTEGRATION:

- Works with existing Mingus user tracking

- Respects user privacy and GDPR compliance

- Real-time updates for key metrics

- API endpoints for admin dashboard

- Alert system for unusual patterns

DATA VISUALIZATION:

- Charts for engagement trends

- Heatmaps for content consumption patterns

- Funnel analysis for user journeys

- Cohort analysis for retention impact

Generate analytics service that provides actionable insights for content strategy and user experience optimization.

**Prompt 15: Admin Dashboard Component**

Create `frontend/admin/ArticleAnalyticsDashboard.jsx` for Mingus admin panel that:

CONTEXT: Admin dashboard for monitoring article library performance and user engagement. Helps optimize content strategy for African American professional audience.

DASHBOARD SECTIONS:

1. Key metrics overview (total articles, active users, engagement rates)

2. Article performance table (sortable by views, completion, ratings)

3. Be-Do-Have phase analytics (user progression, popular phases)

4. Search analytics (popular queries, success rates, failed searches)

5. User assessment trends (score improvements, completion rates)

6. Content recommendations (gaps to fill, trending topics)

VISUALIZATION COMPONENTS:

- Line charts for engagement trends over time

- Bar charts for phase popularity and completion rates

- Pie charts for user distribution across difficulty levels

- Heatmap for reading patterns by time of day/week

- Progress indicators for key metrics vs targets

INTERACTIVE FEATURES:

- Date range selection for time-based analysis

- Filter by user segments (income level, assessment scores)

- Drill-down capability from overview to detailed views

- Export functionality for reports

- Real-time updates for current metrics

PERFORMANCE INSIGHTS:

- Top performing articles with actionable insights

- Underperforming content identification

- User journey analysis (Be → Do → Have progression)

- Conversion metrics (reading to subscription retention)

- A/B testing results display

ADMIN ACTIONS:

- Article promotion/featuring capabilities

- Content moderation and updates

- User segment targeting for recommendations

- Manual content categorization adjustments

Generate comprehensive admin dashboard using React and data visualization libraries (Chart.js or Recharts) that provides clear insights for business decisions.

**🔧 CONFIGURATION & DEPLOYMENT**

**Prompt 16: Environment Configuration**

Create complete environment configuration for Mingus article library feature:

CONTEXT: Mingus Flask app with PostgreSQL database needs configuration for OpenAI integration, article processing, and performance optimization.

CONFIGURATION FILES NEEDED:

1. Updated .env.example with new variables

2. config.py updates for article library settings

3. requirements.txt additions for new dependencies

4. Docker configuration updates (if applicable)

5. Database connection settings optimization

ENVIRONMENT VARIABLES:

- OPENAI\_API\_KEY for article classification

- ARTICLE\_SCRAPING\_RATE\_LIMIT (default: 2 seconds)

- SEARCH\_RESULTS\_PER\_PAGE (default: 20)

- ASSESSMENT\_CACHE\_TIMEOUT (default: 3600 seconds)

- ARTICLE\_CONTENT\_MAX\_LENGTH (default: 10000 chars)

- ENABLE\_ARTICLE\_ANALYTICS (default: True)

PERFORMANCE SETTINGS:

- Database connection pooling for article queries

- Redis cache configuration for search results

- File upload limits for article content

- API rate limiting for search endpoints

- Background task queue for article processing

SECURITY SETTINGS:

- Content Security Policy updates for article display

- CORS settings for article sharing features

- Input validation rules for search queries

- User permission levels for article access

MONITORING CONFIGURATION:

- Logging levels for article operations

- Error tracking for article processing

- Performance monitoring for search queries

- Analytics data retention policies

Generate complete configuration setup that integrates smoothly with existing Mingus infrastructure while supporting the new article library features.

These prompts are designed to be pasted directly into Cursor AI to generate the complete Mingus article library feature. Each prompt provides specific context about the Mingus application and clear requirements for the code to be generated.