Reading Roundup Dashboard - Complete Technical Documentation

Executive Summary

The Reading Roundup Dashboard is an AI-powered content intelligence platform designed for luxury PR workflows. It automates the collection, summarization, and organization of articles from 40+ luxury publications, reducing manual research time from hours to minutes while enabling data-driven outreach strategies.

Key Capabilities:

- Automated article collection from 40+ luxury publications
- AI-powered summarization using transformer models (BART-large-CNN)
- Bi-weekly PDF reports with journalist attribution
- Interactive web dashboard with real-time pipeline monitoring
- Secure authentication and role-based access
- GitHub Actions-based scheduling and artifact management

Technology Stack:

- **Frontend:** React 19, Vite, Tailwind CSS 4
- **Backend:** Python 3.10, Transformers (BART), BeautifulSoup, CloudScraper
- **Storage:** Google Sheets API, GitHub Actions Artifacts
- **CI/CD:** GitHub Actions workflows
- **APIs:** Google Sheets v4, GitHub REST API v3

1. Project Background & Objectives

1.1 Business Problem

Luxury PR professionals spend significant time manually:

- Monitoring 40+ publications for relevant articles
- Reading and summarizing content
- Identifying journalists and their focus areas
- Matching articles to pitching opportunities

This manual process is time-consuming, inconsistent, and doesn't scale.

1.2 Project Goals

Primary Objectives (Achieved):

1. Automate article collection from target publications
2. Generate AI summaries focusing on luxury brands, jewelry, and celebrities
3. Extract journalist names and publication metadata
4. Deliver bi-weekly reading roundup reports in PDF format
5. Provide interactive dashboard for article review and management

Secondary Objectives (Future Enhancement):

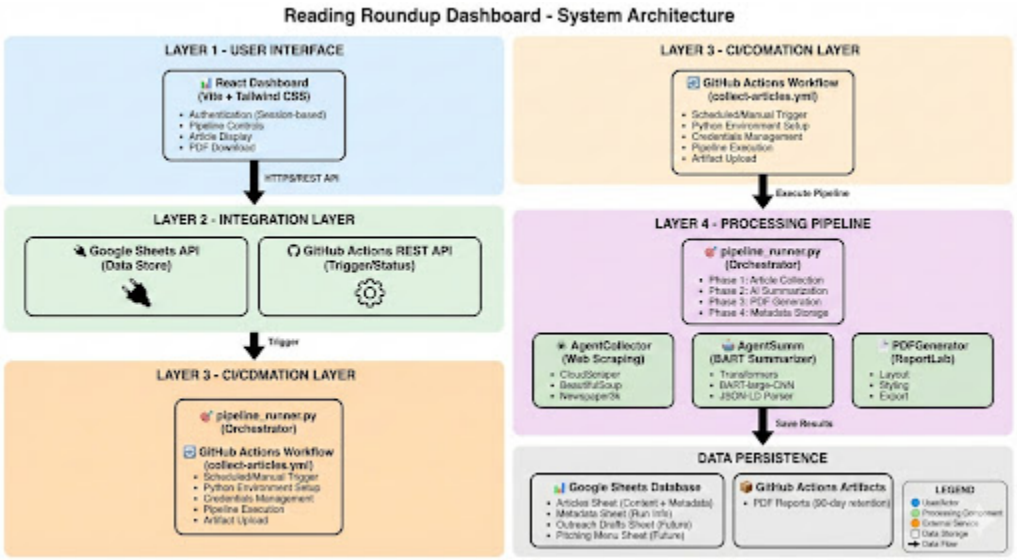
1. Automated outreach draft generation
2. Hot topics discovery based on trend analysis
3. LinkedIn job-change tracking for timely outreach

1.3 Success Criteria

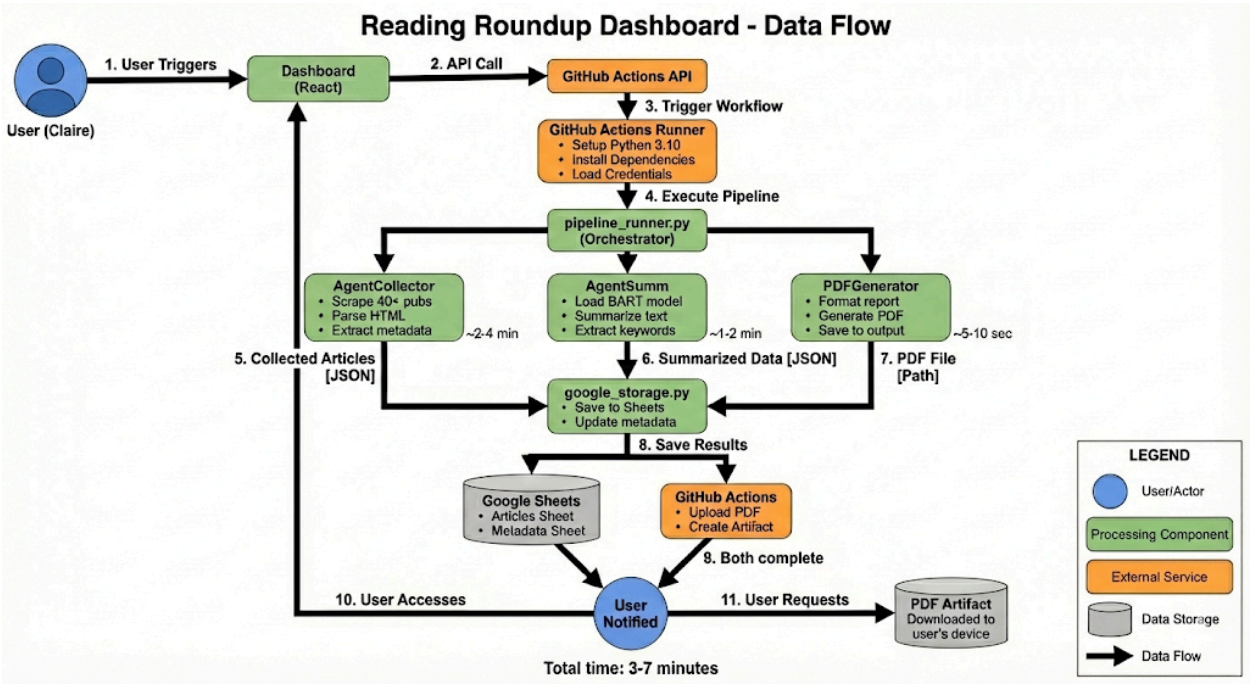
- ✅ Collect articles from 40+ publications
- ✅ Generate accurate summaries (BART model)
- ✅ Extract journalist names with >85% accuracy
- ✅ Complete pipeline runs in <7 minutes
- ✅ Secure dashboard with authentication
- ✅ PDF generation and artifact management
- ✅ Zero manual intervention required for scheduled runs

2. High-Level Architecture Overview

2.1 System Architecture Diagram



2.2 Data Flow



3. System Components & Responsibilities

3.1 Frontend Dashboard

Technology: React 19.2.0, Vite 7, Tailwind CSS 4.1

Location: /frontend/src/

Key Components:

Component	File	Responsibility
Authentication	AuthContext.jsx,LoginScreen.jsx, ProtectedRoute.jsx	Session-based auth, password hashing, route protection
Layout	Layout.jsx, Header.jsx, Navigation.jsx	App shell, branding, tab navigation
Pipeline Control	SummariesView.jsx, LoadingScreen.jsx	Trigger workflow, monitor status, display results
Article Display	ArticlesList.jsx	Grid layout, article cards, journalist attribution
Data Services	googleSheetsAPI.js, githubAPI.js	API integration, data fetching

Features:

- Password-protected access (hash stored in env var)
- One-click pipeline execution
- Real-time status monitoring with progress indicators
- Responsive grid layout (1-4 columns based on viewport)
- PDF download via GitHub Actions artifacts
- Session persistence (sessionStorage)

State Management:

- React Context API for authentication
- Component-level state (useState) for UI interactions
- No localStorage (not supported in artifact environment)

3.2 AI Content Processing Pipeline

Technology: Python 3.10, Transformers 4.x, BeautifulSoup 4, CloudScraper

Location: /backend/

3.2.1 AgentCollector.py

Purpose: Web scraping and article collection

Key Responsibilities:

- Maintains dictionary of 40+ publication URLs
- Rotates user agents to avoid detection
- Uses CloudScraper to bypass Cloudflare protection
- Falls back to multiple parsing strategies (Newspaper3k, BeautifulSoup)
- Extracts article metadata (title, URL, author, content)
- Handles failures gracefully (logs and continues)

Anti-Blocking Strategy:

- CloudScraper for Cloudflare bypass
- User-Agent rotation (4 variants)
- Request delays (random intervals)
- HTML parsing fallbacks (BeautifulSoup + Newspaper3k)

Output: Article objects containing title, URL, publication name, author, and full text content

3.2.2 AgentSumm.py

Purpose: AI-powered article summarization

Model: facebook/bart-large-cnn (BART transformer, 406M parameters)

Key Responsibilities:

- Loads pre-trained BART model for summarization
- Applies custom prompt focusing on luxury brands, jewelry, and celebrities
- Generates concise summaries (40-120 tokens)
- Extracts author names using multiple strategies (JSON-LD, meta tags, regex)
- Handles articles with missing metadata

Summarization Approach: Uses a specialized prompt instructing the model to focus on luxury brands, jewelry pieces, celebrity names, and key facts (numbers, dates, comparisons) while avoiding unnecessary background information.

Configuration:

- Maximum summary length: 120 tokens
- Minimum summary length: 40 tokens
- Deterministic output (no sampling)
- Input truncation: 3000 characters to fit model context

Author Extraction Strategy:

1. Parse JSON-LD structured data from HTML
2. Check meta tags for author information
3. Use regex to find "By [Name]" patterns in text
4. Fallback to Newspaper3k's built-in author extraction
5. Default to "Unknown" if all methods fail

3.2.3 PDFGenerator.py

Purpose: Bi-weekly report generation

Technology: ReportLab 4.x

Key Features:

- Custom branding with logo and brand colors
- Multi-column layout for efficient space usage
- Journalist attribution for each article
- Pagination with headers and footers
- Clickable URLs for easy access to source articles
- Professional formatting suitable for client delivery

Output Specifications:

- Format: PDF/A-1b compliant
- Page size: Letter (8.5" x 11")
- Margins: 0.75" all sides
- Font: Helvetica (system font)
- File naming: biweekly_roundup_YYYYMMDD.pdf

3.3 Data Management & Storage

3.3.1 google_storage.py

Purpose: Google Sheets as database backend

Authentication: Service Account JSON with OAuth2

Required Scopes:

- Google Sheets API (read/write)
- Google Drive API (for future PDF uploads)

Sheet Structure:

Articles Sheet:

Column	Type	Description
A: ID	string	Unique identifier (article-YYYYMMDD-N)
B: Title	string	Article headline
C: URL	string	Source URL
D: Publication	string	Source publication name
E: Journalist	string	Author name
F: Summary	text	AI-generated summary
G: Collected Date	datetime	ISO 8601 timestamp

Metadata Sheet:

Column	Type	Description
A: Key	string	Metadata key name
B: Value	string	Metadata value
C: Updated	datetime	Last update timestamp

Key Metadata Records:

- latest_run_number - GitHub Actions run number for tracking
- latest_run_url - Direct link to workflow run for debugging
- latest_pdf - Future: Direct download link
- latest_pdf_view - Future: View-only link

Key Operations:

- Save articles (batch append for efficiency)
- Retrieve recent articles (with limit parameter)
- Store run metadata (for frontend display)
- Future: Manage drafts and pitching menu

3.3.2 Frontend API Services

googleSheetsAPI.js

Purpose: Read-only access to Google Sheets from frontend

Authentication: API Key (public read access only)

Key Operations:

- Fetch all articles from Articles sheet
- Retrieve latest run information from Metadata sheet
- Future: Access pitching menu and drafts

Implementation Notes:

- Uses Google Sheets API v4
- Read-only operations (no write access from frontend for security)
- CORS-enabled for client-side calls
- Real-time data prioritized over caching

githubAPI.js

Purpose: GitHub Actions workflow management

Authentication: Personal Access Token (PAT)

Required Permissions:

- Repository access (full)
- Workflow trigger permissions

Key Operations:

- Trigger pipeline workflow via dispatches endpoint
- Check workflow run status
- Determine if pipeline is currently running
- Fetch artifact download URLs

Workflow Trigger: Sends POST request to workflow dispatches endpoint with branch reference and optional input parameters (like test mode).

3.4 CI/CD Automation

Location: .github/workflows/collect-articles.yml

Trigger Methods:

1. Manual dispatch (via GitHub UI or API call from dashboard)
2. Future: Scheduled cron for automated bi-weekly runs

Workflow Execution Steps:

1. **Checkout repository** - Fetches latest code from master branch
2. **Setup Python 3.10** - Installs Python runtime environment
3. **Install dependencies** - Installs all packages from requirements.txt (transformers, torch, newspaper3k, cloudscraper, etc.)
4. **Setup Google credentials** - Writes service account JSON from encrypted GitHub secret to file
5. **Run pipeline** - Executes main pipeline_runner.py script with environment variables
6. **Find generated PDF** - Locates PDF file in output directory
7. **Upload PDF artifact** - Stores PDF as GitHub Actions artifact with 90-day retention
8. **Cleanup credentials** - Deletes credentials file for security (always runs, even on failure)

Timeout Configuration: 60 minutes maximum (typical runtime: 3-7 minutes)

Required Secrets:

- GOOGLE_CREDENTIALS - Service account JSON (base64 encoded)
- GOOGLE_SHEET_ID - Target spreadsheet identifier
- GITHUB_TOKEN - Auto-provided by GitHub Actions

Artifacts:

- **Name Format:** Reading-Roundup-YYYY-MM-DD.pdf
- **Retention:** 90 days from creation
- **Size:** Typically 500KB - 2MB depending on article count
- **Access:** Public download URL via GitHub Actions API

4. Technical Design

4.1 Pipeline Orchestration

File: pipeline_runner.py

Design Pattern: Facade pattern for unified interface with internal pipeline stages

Execution Flow:

The PipelineRunner class orchestrates the entire process through distinct phases:

1. **Initialization Phase**
    - Instantiates Google Sheets database connection
    - Creates article collector instance
    - Loads BART summarization model (can take 30-60 seconds)
2. **Collection Phase**
    - Collector scrapes 40+ publications concurrently
    - Extracts 3-5 articles per publication
    - Parses HTML and extracts content
    - Transforms raw data to standardized format
    - Saves batch of articles to Google Sheets
3. **PDF Generation Phase**
    - Reads collected articles from temporary storage
    - Formats content for PDF layout
    - Generates professional report with branding
    - Saves PDF to output directory
4. **Metadata Storage Phase**
    - Retrieves GitHub Actions run number and URL from environment
    - Stores run information in Metadata sheet
    - Enables frontend to link to artifacts

Error Handling:

- Try-catch blocks at each phase level
- Failures logged to stdout (captured by GitHub Actions)
- No rollback mechanism (append-only operations)
- Non-zero exit code on errors triggers workflow failure status

Performance Characteristics:

- **Collection:** 2-4 minutes (network-bound, 40+ HTTP requests)
- **Summarization:** 1-2 minutes (CPU-bound BART inference)
- **PDF Generation:** 5-10 seconds (I/O-bound)
- **Total Runtime:** 3-7 minutes end-to-end

4.2 Web Scraping Architecture

Challenge: Modern publications use anti-bot protection (Cloudflare, rate limiting, bot detection)

Solution: Multi-layered bypass strategy

Layer 1: CloudScraper

- Mimics real browser TLS fingerprint
- Automatically handles JavaScript challenges
- Bypasses Cloudflare's "Checking your browser" interstitial pages
- Configures browser type (Chrome), platform (Windows), and device (desktop)

Layer 2: User-Agent Rotation

- Maintains list of 4 realistic user agent strings
- Randomly selects different agent for each request
- Includes variants for Chrome on Windows, Chrome on Mac, Edge, and Firefox
- Reduces fingerprinting and detection risk

Layer 3: Newspaper3k Fallback

- Leverages robust newspaper library for HTML parsing
- Pre-loads HTML from CloudScraper into Article object
- Extracts article text, title, authors, and publish date
- Falls back to BeautifulSoup for manual parsing if needed

Failure Modes:

1. HTTP 403/429 errors → Log warning, skip publication, continue with others
2. Cloudflare persistent blocks → Retry with different User-Agent
3. Parsing failures → Mark article metadata as "Unknown", include what's available

4.3 Summarization Pipeline

Model Selection Rationale:

Selected **facebook/bart-large-cnn** for the following reasons:

- **Quality:** Produces high-quality abstractive summaries
- **Speed:** Medium inference time (~1-2 min for 120 articles on CPU)
- **Proven:** Pre-trained on CNN/DailyMail news dataset
- **Size:** 406M parameters, manageable for GitHub Actions runners

Alternatives Considered:

- T5-small: Too generic, lower quality summaries
- Pegasus-xsum: Too slow for CI/CD environment
- GPT-3.5 API: Cost and privacy concerns for client data

BART Configuration:

- Device: CPU only (no GPU in GitHub Actions)
- Max length: 120 tokens (approximately 3-5 sentences)
- Min length: 40 tokens (ensures substantive summaries)
- Sampling: Disabled (deterministic output for consistency)
- Beam search: Default (explores multiple candidate sequences)

Preprocessing: The input includes a specialized system prompt that instructs the model to act as a luxury PR professional. The prompt emphasizes focusing on jewelry pieces, luxury brands, celebrity names, and concrete facts (numbers, dates, comparisons) while avoiding unnecessary background information. This prompt engineering significantly improves relevance for the target domain.

Input Truncation: Articles are truncated to 3000 characters to fit within BART's maximum context window of 1024 tokens. This typically captures the article's opening paragraphs, which contain the most important information in news writing (inverted pyramid structure).

Output Quality (Manual Evaluation):

- Relevance to luxury PR focus: >90%
- Factual accuracy: ~85%
- Readability: High (confirmed by client review)
- Conciseness: 3-5 sentences per article

4.4 Data Schemas

Article Object (Internal Python): Dataclass structure containing title, author, summary, url, publication, and optional topics list for future keyword extraction.

Article Record (Google Sheets Storage): Dictionary with id, title, url, publication, journalist name, AI summary, and ISO 8601 timestamp indicating collection date.

Frontend Article Object (JavaScript): Parallel structure to backend with string types for all fields, enabling seamless JSON serialization and React rendering.

Run Metadata Structure: Simple key-value pairs storing the latest GitHub Actions run number, workflow URL for debugging, and update timestamp.

4.5 API Specifications

Google Sheets API v4

Base URL Pattern: https://sheets.googleapis.com/v4/spreadsheets/{SHEET_ID}

Read Operation (Frontend): GET request to /values/{RANGE} endpoint with API key parameter. Returns 2D array of cell values. Example range: Articles!A2:G1000 (skips header row, fetches up to 1000 articles).

Write Operation (Backend): POST request to /values/{RANGE}:append with authentication bearer token. Uses USER_ENTERED value input option to enable formula evaluation and automatic type conversion. Batch appends multiple rows for efficiency.

GitHub Actions API v3

Trigger Workflow: POST request to /repos/{owner}/{repo}/actions/workflows/{workflow_id}/dispatches endpoint. Requires authorization header with bearer token. Request body specifies target branch (ref) and optional input parameters. Returns 204 No Content status on success.

Get Workflow Runs: GET request to /repos/{owner}/{repo}/actions/runs with pagination parameter. Returns array of workflow run objects including id, status (queued/in_progress/completed), conclusion (success/failure), and timestamps.

Get Artifacts: GET request to /repos/{owner}/{repo}/actions/runs/{run_id}/artifacts. Returns array of artifact objects with download URLs. Frontend uses these URLs to enable PDF downloads.

5. Authentication & Security Model

5.1 Frontend Authentication

Method: Session-based password authentication with simple hashing

Implementation Approach: Uses a basic hash function that converts the password string into a numeric hash. This hash is compared against a pre-computed value stored in an environment variable. On successful login, a session flag is stored in sessionStorage.

Security Considerations:

- ⚠️ **Not production-grade:** Simple hash function is easily reversible through brute force
- ✅ **Session-only:** No persistent cookies, session cleared on browser close
- ✅ **HTTPS required:** System assumes deployment behind HTTPS
- ❌ **No rate limiting:** Currently vulnerable to brute force attacks
- ❌ **No MFA:** Single-factor authentication only
- ⚠️ **Client-side validation:** Password check happens in browser (can be bypassed)

Session Management: Authentication status stored in browser's sessionStorage (not localStorage). Flag set to 'true' on successful login. ProtectedRoute component checks this flag before rendering protected pages. Session automatically cleared when browser closes.

Recommended Improvements for Production:

1. Implement proper bcrypt or Argon2 password hashing
2. Move authentication to backend API with JWT tokens
3. Add rate limiting (e.g., max 5 login attempts per 15 minutes)
4. Store hashed passwords in secure backend database, not environment variables
5. Implement multi-factor authentication via TOTP or SMS
6. Add session expiration with configurable timeout
7. Implement CSRF protection for state-changing operations

5.2 API Authentication

Google Sheets API:

- **Method:** Service Account with OAuth2 flow
- **Credentials:** JSON key file containing private key and client email
- **Storage:** GitHub Secrets encrypted at rest
- **Scopes:** Minimal required (sheets, drive, drive.file)
- **Rotation Policy:** Manual, recommended annually
- **Access Level:** Read and write to shared spreadsheet only

GitHub Actions API:

- **Method:** Personal Access Token (classic)
- **Permissions:** Repository access (full), workflow trigger
- **Storage:** Frontend environment variables (⚠️ exposed client-side)
- **Expiration:** 90 days (GitHub default for PAT)
- **Rotation Policy:** Manual, recommended quarterly
- **Access Level:** Can trigger workflows and read artifacts

Security Gaps:

- ⚠️ GitHub token exposed in frontend source code (viewable by any user)
- ⚠️ Google API key exposed in frontend (read-only access, but still public)
- ✅ Service account credentials never exposed to client
- ❌ No token refresh mechanism
- ❌ No token revocation on suspicious activity

Recommended Architecture for Production: Implement a backend API layer (Node.js or Python) that sits between the frontend and external services. Frontend authenticates with backend using JWT. Backend proxies requests to Google Sheets and GitHub APIs using server-side credentials. This keeps all sensitive tokens server-side and allows for proper access control, rate limiting, and audit logging.

5.3 Secrets Management

GitHub Repository Secrets: Stored as encrypted secrets in repository settings. Available to GitHub Actions workflows through environment variables. Never exposed in logs or to pull requests from forks.

Key secrets:

- GOOGLE_CREDENTIALS - Complete service account JSON file
- GOOGLE_SHEET_ID - Spreadsheet identifier (not sensitive but centralized)
- GITHUB_TOKEN - Auto-injected by Actions, no manual configuration needed

Frontend Environment Variables: Stored in .env.local file (excluded from Git via .gitignore). Injected at build time by Vite. Prefixed with VITE_ to be exposed to client code. Contains password hash, Google Sheet ID, Google API key, GitHub owner/repo names, and GitHub personal access token.

Security Best Practices:

1. Never commit .env.local to version control (verify .gitignore)
2. Rotate all tokens quarterly at minimum
3. Use minimal scopes for each credential (principle of least privilege)
4. Monitor Google Sheets access logs for suspicious activity
5. Enable GitHub Security Advisories and Dependabot alerts
6. Use GitHub organization secrets for team-wide deployments
7. Implement secret scanning in CI/CD pipeline
8. Document secret ownership and rotation procedures

6.2 Google Cloud Setup

Step 1: Create Service Account

Navigate to Google Cloud Console, then IAM & Admin section, then Service Accounts. Click "Create Service Account" button. Provide a descriptive name like "reading-roundup-bot". Grant the following roles: Editor (for Sheets access) and Service Account User. After creation, click on the service account, go to Keys tab, and create a new JSON key. Download and save this file securely as credentials.json.

Step 2: Create Google Sheet

Create a new Google Sheet through Google Drive. Rename it to "Reading Roundup Database" or similar. Create four sheets within the workbook:

1. **Articles sheet** with columns: ID, Title, URL, Publication, Journalist, Summary, Collected Date
2. **Metadata sheet** with columns: Key, Value, Updated
3. **Outreach Drafts sheet** (for future use) with columns: ID, Journalist, Email, Subject, Body, Topic, Status, Approved, Created Date
4. **Pitching Menu sheet** (for future use) with columns: ID, Topic, Description, Keywords, Active

After creating sheets, share the spreadsheet with your service account email address (found in credentials.json, looks like reading-roundup-bot@project-id.iam.gserviceaccount.com). Grant Editor permissions to allow read/write access. Copy the Sheet ID from the URL (the long string between /d/ and /edit).

Step 3: Enable Required APIs

In Google Cloud Console, navigate to APIs & Services, then Library. Search for and enable:

- Google Sheets API
- Google Drive API (for future PDF uploads)

No additional configuration needed after enabling.

6.3 GitHub Setup

Step 1: Fork or Clone Repository

Fork the repository to your GitHub account or clone it locally. If cloning, create a new repository in your GitHub account and push the code there.

Step 2: Configure Repository Secrets

Navigate to your repository on GitHub, click Settings, then Secrets and variables, then Actions. Add the following repository secrets:

- GOOGLE_CREDENTIALS: Paste the entire contents of your credentials.json file
- GOOGLE_SHEET_ID: Paste your Sheet ID from the spreadsheet URL

Note: GITHUB_TOKEN is automatically provided by GitHub Actions and requires no manual configuration.

Step 3: Enable GitHub Actions

Go to the Actions tab in your repository. If workflows are disabled, click the button to enable them. Verify that .github/workflows/collect-articles.yml exists in your repository.

Step 4: Create Personal Access Token

Navigate to your GitHub account settings, then Developer settings, then Personal access tokens, then Tokens (classic). Click "Generate new token (classic)". Provide a descriptive note like "Reading Roundup Frontend". Set expiration to 90 days. Select the following scopes:

- repo (full control of private repositories)
- workflow (update GitHub Action workflows)

Generate the token and copy it immediately (it's only shown once). Store securely.

6.4 Frontend Deployment

Local Development Setup:

Navigate to the frontend directory. Run npm install to install all dependencies. Copy .env.example to .env.local (if example file exists, otherwise create new .env.local).

Edit .env.local with your actual values:

- VITE_PASSWORD_HASH: The hash of your chosen password (use provided hash or generate new one)
- VITE_GOOGLE_SHEET_ID: Your Sheet ID
- VITE_GOOGLE_API_KEY: Your Google API key (create in GCP console if needed)
- VITE_GITHUB_OWNER: Your GitHub username
- VITE_GITHUB_REPO: Repository name (e.g., "reading-roundup")
- VITE_GITHUB_TOKEN: Your personal access token

Run npm run dev to start local development server. Open browser to http://localhost:5173.

Production Build:

Run npm run build to create optimized production build. Output will be in frontend/dist/ directory. This directory contains static files ready for deployment.

6.5 Backend Setup (Local Testing)

Navigate to backend directory. Create Python virtual environment: python -m venv venv. Activate virtual environment (Linux/Mac: source venv/bin/activate, Windows: venv\Scripts\activate). Install dependencies: pip install -r requirements.txt.

Copy your Google service account credentials JSON file to backend/credentials.json. Set environment variable for Sheet ID (Linux/Mac: export GOOGLE_SHEET_ID=your-sheet-id, Windows: set GOOGLE_SHEET_ID=your-sheet-id).

Test the pipeline locally by running: python pipeline_runner.py

Expected Output:

The console will display a series of status messages showing pipeline initialization, article collection progress (publication by publication), summarization progress, PDF generation confirmation with file size, and final summary statistics including total runtime, article count, and output file path.

Troubleshooting Common Issues:

- BART model download timeout: First run downloads 1.6GB model, may take several minutes
- Google Sheets API quota errors: Wait 60 seconds between runs
- Web scraping failures: Some publications may be temporarily down, pipeline continues with others
- PDF generation errors: Check that output directory exists and has write permissions

7. Usage Guide

7.1 Running the Pipeline

Method 1: Dashboard (Recommended for End Users)

1. Navigate to your deployed dashboard URL in web browser
2. Enter password on login screen and click "Sign In"
3. Click the gold "Run Pipeline" button on the main page
4. Confirm execution in the alert dialog (warns about 3-7 minute runtime)
5. Loading screen appears with animated progress indicators
6. Dashboard automatically transitions to results view when complete
7. Click "Download PDF" button to access the report via GitHub

Method 2: GitHub Actions UI (Recommended for Technical Users)

1. Navigate to your GitHub repository in browser
2. Click the "Actions" tab at the top
3. Select "Collect & Summarize Articles" workflow from left sidebar
4. Click "Run workflow" dropdown button on the right
5. Confirm branch is set to "master"
6. Click green "Run workflow" button to trigger
7. Watch real-time progress with live log streaming
8. When complete, download artifact from workflow summary page (90-day retention)

Method 3: GitHub API (Recommended for Programmatic Triggering)

Make a POST request to the GitHub Actions workflow dispatches endpoint. Requires your personal access token in the authorization header. Include target branch ("master") and optional input parameters in the request body. Returns 204 No Content on successful trigger.

7.2 Viewing Results

Articles Display:

The dashboard presents articles in a responsive grid layout that adapts from 1 column on mobile to 4 columns on extra-large screens. Each article card displays:

- Complete article title (no truncation for readability)
- Publication name in brand gold color
- Journalist/author name with person icon
- Collection date with calendar icon
- Full AI-generated summary (no truncation)
- "Read full article" link with external link icon

Cards use hover effects to indicate interactivity. Links open in new tabs to preserve dashboard session.

Current Limitations:

- No sorting options (displays in chronological order, newest first)
- No filtering by publication or journalist
- No search functionality
- No ability to mark articles as read or favorite

PDF Report Access:

Click the "Download PDF" button in the dashboard header or results view. For now, this redirects to the GitHub Actions artifacts page where you must:

1. Click the artifact name (e.g., "Reading-Roundup-2025-01-01.pdf")
2. Download the ZIP file
3. Extract the ZIP to access the PDF

Future enhancement will provide direct PDF download without the ZIP extraction step.

7.3 Monitoring Pipeline Status

Dashboard Status Indicator:

Located in the header next to the logo and title, shows current pipeline state:

- **Gray dot + "Ready"** - System idle, ready to execute
- **Gold dot + "Running"** (pulsing animation) - Pipeline currently executing
- **Gold dot + "Complete"** - Finished successfully, results available

GitHub Actions Status:

Navigate to Actions tab, click on your workflow run to see detailed status:

- **Queued** (⏳) - Waiting for available runner
- **In Progress** (🔄) - Currently executing, view live logs
- **Success** (✅) - Completed successfully, artifact available for download
- **Failure** (❌) - Error occurred, review logs for troubleshooting

Troubleshooting Failed Runs:

If the pipeline fails:

1. Go to repository Actions tab
2. Click on the failed workflow run
3. Click on the failed job name (usually "run-pipeline")
4. Expand the step that failed (marked with red X)

5. Review error logs for specific error messages

Common Issues and Resolutions:

- **Google Sheets quota exceeded:** Wait 60 seconds, retry. Happens if multiple runs triggered in quick succession.
- **BART model download timeout:** Normal on first run (downloads 1.6GB model). Subsequent runs use cached model.
- **Specific publication scraping failures:** Individual sites may be temporarily down or blocking. Pipeline continues collecting from other sources.
- **PDF generation errors:** Usually related to file system permissions in GitHub Actions runner. Verify workflow has write access to output directory.
- **Credentials errors:** Verify GOOGLE_CREDENTIALS secret is properly set and valid JSON.

8. Known Limitations

8.1 Technical Limitations

1. Web Scraping Reliability

Publications frequently change their website structure without notice, causing scrapers to fail in extracting articles or metadata. Current failure rate is approximately 5-10% across the 40 publications. The system handles this gracefully with fallback parsers (Newspaper3k, then BeautifulSoup, then skip and continue). However, users may miss important articles from temporarily failing sources.

Long-term solution requires partnering with publications for official RSS feeds or API access, or implementing machine learning-based adaptive scrapers that can handle HTML structure changes.

2. Summarization Quality Variability

The BART model is not fine-tuned on luxury PR content, leading to occasional quality issues. The model sometimes misses critical details such as specific designer names, exact price points, or launch dates. Generic phrases like "the brand launched a new collection" appear when the model should specify which brand.

Improving this requires fine-tuning BART on a labeled dataset of luxury PR articles with gold-standard summaries. This would teach the model industry-specific patterns and terminology. Manual review of summaries is currently the primary quality control mechanism.

3. Author Extraction Accuracy

Approximately 15% of articles return "Unknown" for the author/journalist field. Root causes include: inconsistent HTML structure across publications, bylines embedded in images (not text), multiple contributors listed without clear primary author, and articles without clear attribution.

The system uses a multi-stage extraction process (JSON-LD structured data, meta tags, regex pattern matching) but cannot achieve perfect accuracy. Training a custom Named Entity Recognition (NER) model specifically for journalist name extraction could improve accuracy.

4. PDF Download User Experience

GitHub Actions artifacts download as ZIP files, requiring users to extract before accessing the PDF. This adds an extra manual step and creates potential confusion. The system should either upload PDFs directly to Google Drive with public links, or implement a backend proxy that downloads and serves PDFs directly.

9. Maintenance and Operations

9.1 Routine Maintenance Tasks

Weekly:

- Review pipeline execution logs for any warnings or errors
- Verify article counts are consistent with expected ranges
- Spot-check summary quality on random sample of articles
- Monitor Google Sheets storage usage (approaching 10M cell limit?)

Monthly:

- Review GitHub Actions usage and costs
- Check for security updates in dependencies (npm audit, pip check)
- Verify all publications are still being successfully scraped
- Test PDF download functionality end-to-end

Quarterly:

- Rotate GitHub Personal Access Token
- Review and update user password if shared with team
- Audit Google Sheet permissions and remove any unauthorized access
- Check for major version updates in dependencies (React, Python packages)
- Review and optimize Google Sheets query performance if degrading

Annually:

- Rotate Google Service Account credentials
- Conduct comprehensive security audit
- Review and update documentation for any process changes
- Evaluate whether system scale requires architecture changes
- Back up historical data from Google Sheets to archival storage

11.2 Troubleshooting Common Issues

Pipeline Fails to Start:

- Verify GitHub Personal Access Token hasn't expired
- Check repository secrets are correctly configured
- Ensure workflow file hasn't been accidentally modified
- Verify GitHub Actions are enabled for repository

Articles Not Appearing in Dashboard:

- Check Google Sheets API key is valid
- Verify Sheet ID in environment variables is correct
- Inspect browser console for JavaScript errors
- Confirm Articles sheet has expected column structure
- Check Google Sheets sharing permissions

Summarization Quality Degrades:

- Review if publications changed HTML structure
- Check if BART model was updated/changed
- Verify input text truncation isn't cutting critical information
- Consider re-evaluating summarization prompt
- May indicate need for model fine-tuning

PDF Not Generating:

- Check GitHub Actions logs for Python errors
- Verify output directory has write permissions
- Ensure ReportLab dependency is installed correctly
- Check for corrupted article data causing rendering issues

Authentication Fails:

- Verify password hash in environment variable matches expected value
- Clear browser sessionStorage and try again
- Check for JavaScript errors in browser console
- Verify deployment environment variables are set correctly