NewsShield — Product Requirement Document (PRD) & Basic Requirements

Prepared for: Student Project / Portfolio

1. Executive Summary

NewsShield is a full■stack AI platform that automatically collects news, classifies it into categories, and evaluates the credibility of the content. The system will provide confidence/trust scores for text articles and lay the groundwork for future multimodal support (video/audio verification and deepfake detection).

This document defines the minimum viable product (MVP) features, functional and non■functional requirements, architecture, data and model needs, UI wireframe descriptions, development milestones, testing plans, and deployment instructions for a working production■like system.

2. Scope (MVP vs Future)

MVP (Phase 1): Real-time text news ingestion, topic classification, fake■news detection for text, credibility scoring, searchable dashboard, user accounts, bookmarking and alerts.

Phase 2: Source reputation engine, automated fact■checking by cross■referencing claims, scheduled daily digests, multilingual support.

Phase 3 (Future): Multimodal verification: video/audio deepfake detection, cross■media correlation, blockchain anchoring for verified items.

3. Key Features (MVP)

• News ingestion from multiple sources via NewsAPI, RSS feeds, and optional web scrapers.

• Article preprocessing pipeline (cleaning, tokenization, language detection).

• Topic classification (Politics, Business, Tech, Sports, Health, Entertainment, World, Local).

• Fake news detection for text with a credibility score (0–100%) and label (Likely Real / Likely Fake / Uncertain).

• Source reputation: store and display source trust metadata.

• User dashboard: feed view, filters, search, article details, bookmarking, and report a claim.

• Admin panel: manage sources, view model performance and flagged items.

• Alerts: email or in■app notification for trending fake news or user subscriptions.

• RESTful API endpoints for article retrieval, classification, and verification.

4. Functional Requirements

FR1. The system shall ingest articles every N minutes (configurable) from configured sources.

FR2. The system shall store raw and processed article text, metadata, and model outputs in the database.

FR3. The system shall classify each article into one of the predefined categories with >= target accuracy.

FR4. The system shall provide a credibility score and a short explanation for each prediction (top signals).

FR5. The system shall allow users to search, filter, bookmark, and flag articles.

FR6. The system shall expose a secure REST API for frontend consumption and third■party use.

FR7. The system shall log predictions and user feedback for model retraining.

5. Non-functional Requirements

NFR1. Security: API authentication (JWT), protected admin routes, secure storage of user credentials (bcrypt).

NFR2. Performance: Page load time < 2s for common operations; classification latency < 1s for single article (batch processing allowed).

NFR3. Scalability: Microservice friendly; message queue for ingestion (e.g., RabbitMQ / Kafka).

NFR4. Reliability: 99% availability for core ingestion and classification services (development target).

NFR5. Maintainability: Dockerized services, documentation, and CI pipeline for tests and deployments.

6. System Architecture (High Level)

Components:

• Ingestion Service – scheduler & scrapers (fetches articles)

• Preprocessing Service – text normalization, language detection

• Classification Service – BERT■based topic classifier

• Verification Service – fake■news detector producing credibility score

• Database – MongoDB (primary), Redis (cache), PostgreSQL optional for relational data

• API Gateway / Backend – FastAPI or Flask providing REST endpoints

• Frontend – React SPA

• Admin Service – monitoring and model retraining orchestration

• Message Broker – RabbitMQ or Kafka

• Optional: Worker pool (Celery) for async tasks

Data Flow:

1. Scheduler triggers ingestion → fetches articles → stores raw article

2. Preprocessing cleans text and stores processed text

3. Classification and Verification models consume processed text (sync or via workers)

4. Results are written back to DB and surfaced through API to frontend

7. Data Model (High-level Entities)

Article:

• id, title, body, summary, source, url, publish_date, language, raw_html

• categories (predicted), category_confidence

• credibility_score, credibility_label, explanation_signals

• ingestion_timestamp, processing_timestamp

User:

• id, name, email, password_hash, role (user/admin), preferences, subscriptions

Source:

• id, name, url, reputation_score, first_seen, last_seen

8. ML & Data Requirements

• Datasets: FakeNewsNet, LIAR, Kaggle Fake News Dataset, BBC (for topic labels). Collect in■house labeled samples via user feedback.

• Models (MVP): DistilBERT/BERT fine■tuned for topic classification; BERT/XGBoost ensemble for fake■news detection.

• Features: text embeddings, TF■IDF, sentiment scores, readability, source reputation, headline■body similarity.

• Evaluation: accuracy, precision, recall, F1, ROC■AUC for binary fake detection. Use cross■validation and holdout test sets.

• Explainability: Use SHAP or LIME to surface top features/signals behind predictions.

9. API Spec (examples)

Endpoint	Method	Description
/api/v1/articles	GET	List articles with filters (category, trust range, search)
/api/v1/articles/{id}	GET	Retrieve article details and credibility report
/api/v1/ingest/start	POST	Admin: start/stop ingestion (protected)
/api/v1/auth/login	POST	User login, returns JWT
/api/v1/verify	POST	Submit arbitrary text or URL for classification & verification

10. Frontend Screens / UX Flow

• Home / Feed: Infinite scroll of latest articles with category tags and credibility badge.

• Article view: Full article, source link, category label, credibility score (0–100), short explanation of signals, share/bookmark/report buttons.

• Search & Filters: Filter by category, date range, trust level, source.

• User profile: Saved articles, subscriptions, preferences.

• Admin dashboard: Source management, queue status, model metrics, retrain button.

11. Development Roadmap & Milestones (Suggested)

Week 1–2: Project setup, repo, Docker, basic ingestion from 5 sources, DB schema.

Week 3–4: Preprocessing pipeline, initial topic classifier (baseline), frontend skeleton.

Week 5–6: Fake news detection baseline (XGBoost + TF■IDF), API endpoints, basic UI.

Week 7–8: Model improvements (fine■tune BERT), explainability, bookmarking, user auth.

Week 9–10: Admin dashboard, alerts, testing, documentation, deployment scripts.

Week 11–12: Buffer for bug fixes and optional Phase 2 features (source reputation, scheduled digests).

12. Testing & Evaluation Plan

• Unit tests for ingestion, preprocessing, and APIs (pytest).

• Integration tests for end■to■end flows (ingest → classify → display).

• Model evaluation with holdout test sets and continuous monitoring for concept drift.

• User acceptance testing with sample users; collect feedback and false■positive reports.

13. Deployment

• Containerize services with Docker. Use Docker Compose for local dev and Kubernetes for production.

• CI: GitHub Actions to run tests and build images. CD: Deploy to cloud (AWS/GCP/Azure) using Terraform/Helm.

• Logging & Monitoring: Prometheus + Grafana for metrics; ELK stack for logs.

• Secrets: Vault or cloud provider secret manager for API keys and DB credentials.

14. Security & Privacy

• Use HTTPS for all endpoints. Store passwords with strong hashing (bcrypt) and salt.

• GDPR considerations: Provide a way to delete personal data and export user data.

• Rate limits on public APIs to prevent abuse. Verify third■party source licensing before scraping.

15. Success Criteria

• Functional: Fully working MVP with ingestion, classification, fake detection, and dashboard.

• Model: Topic classification F1 > 0.85 (target) and fake■news detection F1 > 0.75 for initial model.

• User: At least 80% of test users find credibility score helpful (survey).

• Ops: Automated deployment and CI in place; daily ingestion runs without failure.

16. Appendix: Datasets & Tools

Recommended datasets: FakeNewsNet, LIAR, Kaggle Fake News, BBC News Dataset.

Recommended libs & tools: HuggingFace Transformers, spaCy, scikit■learn, XGBoost, SHAP, FastAPI, React, Docker, PostgreSQL/MongoDB.

Sample third■party APIs: NewsAPI.org, GDELT, YouTube Data API (future video verification).

17. Notes & Next Steps

• If you want, I can produce: detailed data schema SQL/NoSQL models, API OpenAPI spec, UI mockups, or a complete project timeline with resource estimates.

• Tell me which deliverable you want next and I will generate it (e.g., OpenAPI spec, database seed script, or PowerPoint presentation).