PulseStream AI – Project Requirements Document

1. Project Overview

PulseStream AI is a real-time news and trend recommendation platform combining **data engineering** and **AI development**. It ingests data from multiple APIs, processes it through a scalable pipeline, stores it in a data warehouse, and delivers personalized recommendations through an AI model served via FastAPI.

---

2. Objectives

- Build an **end-to-end data and AI system** integrating real-time streaming, orchestration, and model serving.
- Demonstrate proficiency in **Kafka**, **Airflow**, **DBT**, **Snowflake**, **Docker**, **FastAPI**, and **Cloud Platforms (Azure/AWS)**.
- Provide a working example of data engineering + AI collaboration.

---

3. Tools & Technologies

Layer	Tools / Technologies
Data Ingestion	Kafka, Python, APIs (News, Twitter, Reddit)
Stream Processing	Kafka Consumer, spaCy, Sentiment Model (API)
Data Storage	Snowflake, Azure Blob / AWS S3
Transformation	DBT, SQL models
Orchestration	Apache Airflow
AI Model	Python (scikit-learn / PyTorch), Recommendation Algorithms
API Serving	FastAPI
Deployment	Docker, Azure Container Apps / AWS ECS
Monitoring	Airflow UI, CloudWatch / Azure Monitor

---

4. System Architecture Summary

Data Flow (Simplified)

1. **Data Ingestion** → APIs → Kafka (raw_news_feed)

2. **Stream Processing** → Kafka Consumer → Data Enrichment → Kafka (cleaned_news_feed)
3. **Storage** → Load to Snowflake tables (Raw → Staging → Curated)
4. **Transformation** → DBT models build analytical tables
5. **Model Training** → Airflow triggers Python ML job → Save model to Blob/S3
6. **Model Serving** → FastAPI serves predictions (recommendations)
7. **Orchestration** → Airflow manages all DAGs and dependencies
8. **Monitoring** → CloudWatch/Azure Monitor tracks logs and performance

---

5. Step-by-Step Implementation Plan

Phase 1: Setup & Ingestion

- Create Kafka cluster (local Docker / managed MSK or Event Hub)
- Write Python producers to fetch live data from News APIs, Reddit, Twitter
- Push events to Kafka topic `raw_news_feed`

Phase 2: Stream Processing & Enrichment

- Develop Kafka consumer in Python to read data from `raw_news_feed`
- Clean and enrich data using:
- **spaCy** (Named Entity Recognition)
- **Sentiment Model** via API (e.g., OpenAI or HuggingFace)
- Publish enriched data to `cleaned_news_feed`

Phase 3: Data Storage & Warehouse

- Integrate Kafka consumer output with **Snowflake** (use Snowpipe or Python connector)
- Create schemas: `raw_data`, `staging`, `analytics`
- Store enriched records in the warehouse for transformation

Phase 4: Transformation with DBT

- Define models for transformations:
- `stg_news_raw` → cleansed version of raw data
- `dim_topics`, `fact_user_activity`, `fact_article_features`
- Test transformations using DBT tests
- Schedule DBT jobs in Airflow

Phase 5: Model Training

- Create Airflow DAG `model_training_dag`
- Train hybrid recommendation model:
- Collaborative Filtering (based on user interactions)
- Content-based (using embeddings from APIs)
- Save model artifact to **Azure Blob / AWS S3**

Phase 6: Model Serving with FastAPI

- Build FastAPI endpoints:
- `/recommendations?user_id=xyz` → returns top N recommendations
- `/feedback` → records user feedback for retraining
- Deploy FastAPI via Docker on **Azure Container Apps / AWS ECS**

Phase 7: Orchestration with Airflow

- Implement DAGs for:
- Data ingestion
- Transformation (DBT)
- Model training & deployment
- Configure Airflow web UI and scheduler for monitoring

Phase 8: Monitoring & Scaling

- Set up monitoring dashboards with CloudWatch / Azure Monitor
- Add logging for ingestion, transformation, and inference
- Implement retry & alert policies in Airflow DAGs

---

6. Example Data Pipeline DAG (Airflow)

```
news_ingestion_dag
 ├── fetch_news_from_APIs
 ├── publish_to_kafka
 ├── consume_and_clean
 ├── load_to_snowflake
 ├── run_dbt_transformations
 ├── retrain_recommendation_model
 └── deploy_latest_model_to_FastAPI
```

---

7. Deliverables

- Dockerized microservices for ingestion, API, and orchestration
- Functional Airflow DAGs for all stages
- DBT repository with modular SQL models
- Snowflake schemas and views
- FastAPI service with working `/recommendations` and `/feedback` routes
- Logs, dashboards, and documentation for monitoring

---

8. Success Criteria

- Data successfully flows from ingestion → processing → warehouse → model → API
- Personalized recommendations refresh dynamically
- Airflow DAGs run without failure and handle retries
- API response latency < 200ms for inference
- Full system runs on Azure or AWS with containerized deployments

---

9. Future Enhancements

- Implement online model updates using real-time feedback
- Add A/B testing for multiple models
- Build a Streamlit or Power BI dashboard for analytics
- Integrate authentication and rate limiting in API

Author: Tanish Hanjra
Version: 1.0
Date: October 2025