News Intelligence App – Full Stack Prototype (Investor + End-User)

This repo is a **complete, runnable prototype**: a FastAPI backend that aggregates news (Google News RSS fallback) and a React + Tailwind frontend with two modes: **Investor View** (dashboard) and **User View** (personalized feed). It's designed to demo well and be extendable to your existing scrapers.

File Tree

```
news-intel/
├─ server/
│  ├─ app/
│  │  ├─ main.py
│  │  ├─ models.py
│  │  ├─ providers/
│  │  │  ├─ __init__.py
│  │  │  ├─ rss_provider.py
│  │  │  └─ adapter_existing.py
│  │  └─ utils.py
│  ├─ requirements.txt
│  ├─ .envmple
│  └─ README.md
└─ client/
   ├─ index.html
   ├─ package.json
   ├─ postcss.config.cjs
   ├─ tailwind.config.cjs
   ├─ tsconfig.json
   ├─ vite.config.ts
   └─ src/
      ├─ main.tsx
      ├─ App.tsx
      ├─ api.ts
      ├─ styles.css
      └─ components/
         ├─ ModeToggle.tsx
         ├─ KeywordChips.tsx
         ├─ ArticleCard.tsx
         └─ TrendsChart.tsx
```

Integration note: The backend first tries to use your existing modules (if present): `news_api_service.py` or `googlenews_scraper_window_version.py`. If not available, it falls back to a **Google News RSS provider**. No API keys required for the RSS path.

Backend (FastAPI)

`server/app/models.py`

```
from __future__ import annotations
from pydantic import BaseModel, Field
from typing import List, Optional
from datetime import datetime

class Article(BaseModel):
    title: str
    url: str
    source: str
    published_at: Optional[datetime] = None
    snippet: Optional[str] = None
    keyword_hits: List[str] = Field(default_factory=list)

class SearchRequest(BaseModel):
    keywords: List[str]
    max_results: int = 50

class SearchResponse(BaseModel):
    total: int
    articles: List[Article]

class TrendPoint(BaseModel):
    keyword: str
    timestamp: datetime
    count: int

class TrendsResponse(BaseModel):
    points: List[TrendPoint]
```

`server/app/utils.py`

```
from __future__ import annotations
from typing import Iterable, List
from datetime import datetime
import re
```

```
TS_FORMATS = [
    "%a, %d %b %Y %H:%M:%S %Z",
    "%a, %d %b %Y %H:%M:%S %z",
    "%Y-%m-%dT%H:%M:%SZ",
    "%Y-%m-%dT%H:%M:%S%z",
]

def parse_ts(value: str | None):
    if not value:
        return None
    for fmt in TS_FORMATS:
        try:
            return datetime.strptime(value, fmt)
        except Exception:
            pass
    return None

_word = re.compile(r"[A-Za-z0-9_]+", re.IGNORECASE)

def keyword_hits(text: str, keywords: Iterable[str]) -> List[str]:
    if not text:
        return []
    text_l = text.lower()
    hits = []
    for k in keywords:
        k_l = k.lower().strip()
        if not k_l:
            continue
        if k_l in text_l:
            hits.append(k)
    return hits
```

server/app/providers/rss_provider.py

```
from __future__ import annotations
from typing import List
import feedparser
from datetime import datetime
from ..models import Article
from ..utils import parse_ts, keyword_hits

# Simple Google News RSS provider (no API key required)
# Example feed: https://news.google.com/rss/search?q=tesla&hl=en-
US&gl=US&ceid=US:en

BASE = "https://news.google.com/rss/search?q={q}&hl=en-US&gl=US&ceid=US:en"
```

```
class RSSProvider:
    def fetch(self, keywords: List[str], max_results: int = 50) ->
List[Article]:
        results: List[Article] = []
        for kw in keywords:
            url = BASE.format(q=kw.replace(" ", "+"))
            feed = feedparser.parse(url)
            for entry in feed.entries:
                ts = parse_ts(getattr(entry, "published", None))
                title = getattr(entry, "title", "").strip()
                source = getattr(entry, "source", {}).get("title") if
getattr(entry, "source", None) else "Google News"
                link = getattr(entry, "link", "")
                snippet = getattr(entry, "summary", None)
                hits = keyword_hits(f"{title} {snippet}", [kw])
                results.append(Article(title=title, url=link, source=source or
"Google News", published_at=ts, snippet=snippet, keyword_hits=hits))
        # Deduplicate by URL, prefer newest
        dedup = {}
        for art in results:
            if art.url in dedup:
                # merge keyword hits
                prev = dedup[art.url]
                prev.keyword_hits = list(sorted(set(prev.keyword_hits +
art.keyword_hits)))
                if (art.published_at or datetime.min) > (prev.published_at or
datetime.min):
                    dedup[art.url] = art
            else:
                dedup[art.url] = art
        items = list(dedup.values())
        items.sort(key=lambda a: a.published_at or datetime.min, reverse=True)
        return items[:max_results]
```

server/app/providers/adapter_existing.py

```
from __future__ import annotations
from typing import List
from ..models import Article
from datetime import datetime

class ExistingAdapter:
    """
    Attempts to call your local modules if present:
      - news_api_service.py: expected function search_news(keywords: List[str],
```

```
max_results: int) -> List[dict]
      - googlenews_scraper_window_version.py: expected function
search_news_window(keywords: List[str], max_results: int) -> List[dict]
    Each dict should include: title, url, source, published_at (ISO), snippet.
    If modules are missing or raise ImportError, this adapter raises
ImportError, and caller should fallback.
    """
    def fetch(self, keywords: List[str], max_results: int = 50) ->
List[Article]:
        try:
            # Try news_api_service first
            import importlib
            try:
                nas = importlib.import_module("news_api_service")
                data = nas.search_news(keywords=keywords,
max_results=max_results)
            except Exception:
                gns =
importlib.import_module("googlenews_scraper_window_version")
                data = gns.search_news_window(keywords=keywords,
max_results=max_results)

            articles: List[Article] = []
            for d in data:
                ts = None
                try:
                    ts = datetime.fromisoformat(d.get("published_at"))
                except Exception:
                    ts = None
                articles.append(Article(
                    title=d.get("title", ""),
                    url=d.get("url", ""),
                    source=d.get("source", "Custom"),
                    published_at=ts,
                    snippet=d.get("snippet"),
                    keyword_hits=d.get("keyword_hits", []),
                ))
            return articles
        except Exception as e:
            raise ImportError(str(e))
```

`server/app/main.py`

```
from __future__ import annotations
from fastapi import FastAPI, Query
from fastapi.middleware.cors import CORSMiddleware
```

```
from typing import List
from datetime import datetime
from .models import SearchResponse, TrendsResponse, TrendPoint
from .providers.rss_provider import RSSProvider
from .providers.adapter_existing import ExistingAdapter

app = FastAPI(title="News Intelligence API", version="0.1.0")

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Provider selection with graceful fallback
try:
    _provider = ExistingAdapter()
    # Smoke test to see if it imports; will fallback to RSS if not
    # (We don't call yet to avoid blocking startup.)
    _existing_ok = True
except Exception:
    _existing_ok = False

_rss = RSSProvider()

@app.get("/api/articles", response_model=SearchResponse)
async def get_articles(keywords: List[str] = Query(default=["tesla", "ai",
"crypto"]), max_results: int = 50):
    if _existing_ok:
        try:
            items = _provider.fetch(keywords, max_results)
        except Exception:
            items = _rss.fetch(keywords, max_results)
    else:
        items = _rss.fetch(keywords, max_results)
    return {"total": len(items), "articles": items}

@app.get("/api/trends", response_model=TrendsResponse)
async def get_trends(keywords: List[str] = Query(default=["tesla", "ai",
"crypto"])):
    # Lightweight trend proxy: counts per keyword in the last 24h based on RSS
    items = _rss.fetch(keywords, max_results=200)
    pts: List[TrendPoint] = []
    now = datetime.utcnow()
    buckets = {}
    for kw in keywords:
```

```
            buckets[kw] = 0
        for a in items:
            for kw in a.keyword_hits:
                buckets[kw] += 1
        for kw, cnt in buckets.items():
            pts.append(TrendPoint(keyword=kw, timestamp=now, count=cnt))
        return {"points": pts}

@app.get("/api/health")
async def health():
    return {"status": "ok"}
```

`server/requirements.txt`

```
fastapi==0.112.0
uvicorn==0.30.5
pydantic==2.8.2
feedparser==6.0.11
```

`server/.env.example`

```
# Reserved for future providers (e.g., NewsAPI, custom keys)
NEWSAPI_KEY=
```

`server/README.md`

````
# Server

## Quickstart
```bash
cd server
python -m venv .venv && source .venv/bin/activate  # Windows: .venv\\Scripts\
\activate
pip install -r requirements.txt
uvicorn app.main:app --reload --port 8000
````

Endpoints

- `GET /api/articles?keywords=tesla&keywords=ai&max_results=50`
- `GET /api/trends?keywords=tesla&keywords=ai`
- `GET /api/health`
````

Integration with your existing modules

If you have `news_api_service.py` or `googlenews_scraper_window_version.py` on `PYTHONPATH` or in the same folder, the API will try those first. It expects:

```
# news_api_service.py
# def search_news(keywords: List[str], max_results: int) -> List[dict]:
#   return [{"title": str, "url": str, "source": str, "published_at": iso_str,
"snippet": str}, ...]
```

If import fails, it falls back to Google News RSS.

````
---

## Frontend (React + Vite + Tailwind + Recharts + Framer Motion)

### `client/package.json`
```json
{
  "name": "news-intel-client",
  "version": "0.1.0",
  "private": true,
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "preview": "vite preview"
  },
  "dependencies": {
    "framer-motion": "^11.3.23",
    "react": "^18.3.1",
    "react-dom": "^18.3.1",
    "recharts": "^2.12.7"
  },
  "devDependencies": {
    "@types/react": "^18.3.7",
    "@types/react-dom": "^18.3.0",
    "autoprefixer": "^10.4.20",
    "postcss": "^8.4.41",
    "tailwindcss": "^3.4.10",
    "typescript": "^5.5.4",
    "vite": "^5.4.2"
  }
}
````
````

client/tailwind.config.cjs

```
/** @type {import('tailwindcss').Config} */
export default {
  content: [
    './index.html',
    './src/
`
```