AI News Agent - Detailed Technical Specification

1. Application Overview

Purpose: An autonomous AI-powered platform that:

1. Aggregates Pakistan news from Google News RSS
2. Processes articles through Groq LLM with tool augmentation
3. Generates platform-optimized social media posts (X, Instagram, Facebook)
4. Provides real-time processing visibility and automation

---

2. Technology Stack

Core Framework

Component	Technology	Version	Purpose
Frontend	Next.js	14.x	React framework with App Router
Language	TypeScript	5.x	Type-safe JavaScript
Styling	Tailwind CSS	3.x	Utility-first CSS
Components	Shadcn/ui	Latest	Pre-built accessible components

Database & Storage

Service	Purpose	Details
Supabase	Primary database	PostgreSQL with real-time subscriptions
Pinecone	Vector database	RAG memory for agent learning

AI & APIs

Service	API Key Env	Purpose
Groq	`GROQ_API_KEY`	LLM inference (GPT-OSS, Llama models)
Jina AI	`JINA_API_KEY`	Article content scraping
Serper	`SERPER_API_KEY`	Google Search & News API
Pinecone	`PINECONE_API_KEY`	Vector similarity search

---

3. Database Schema (Detailed)

Table: `news_items`

Stores all fetched news articles from RSS feeds.

```
CREATE TABLE news_items (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  title TEXT NOT NULL,                    -- Article headline
  link TEXT NOT NULL,                     -- Source URL
  source_name TEXT,                       -- Publisher (e.g., "Dawn", "Geo News")
  image_url TEXT,                         -- Thumbnail image URL
  content_snippet TEXT,                   -- RSS description/snippet
  pub_date TIMESTAMP WITH TIME ZONE,      -- Publication date from RSS
  hash TEXT UNIQUE NOT NULL,              -- SHA-256 hash for deduplication
  is_new BOOLEAN DEFAULT true,            -- Flag for UI highlighting
  is_posted BOOLEAN DEFAULT false,        -- Has been processed by agent
  posted_platforms TEXT[],                -- Array of platforms posted to
  x_post TEXT,                            -- Generated X/Twitter content
  instagram_caption TEXT,                 -- Generated Instagram caption
  facebook_post TEXT,                     -- Generated Facebook content
  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW()
);
```

Table: `feeder_settings`

Single-row table storing feeder configuration.

```
CREATE TABLE feeder_settings (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  refresh_interval INTEGER DEFAULT 900000,  -- Auto-refresh in milliseconds
  is_active BOOLEAN DEFAULT false,          -- Auto-refresh enabled
  max_retention INTEGER DEFAULT 100,        -- Max articles to keep
  freshness_hours INTEGER,                  -- Only fetch news from last X hours
  last_fetch TIMESTAMP WITH TIME ZONE,      -- Last successful RSS fetch
  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW()
);
```

Table: `agent_settings`

Single-row table storing agent configuration.

```sql
CREATE TABLE agent_settings (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  model TEXT DEFAULT 'gpt-oss-120b',        -- LLM model identifier
  batch_size INTEGER DEFAULT 10,            -- Articles per run
  order_direction TEXT DEFAULT 'desc',      -- 'asc' or 'desc' by pub_date
  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW()
);
```

Table: `agent_runs`

Tracks each agent execution session.

```sql
CREATE TABLE agent_runs (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  status TEXT DEFAULT 'running',            -- running/completed/cancelled/failed
  articles_processed INTEGER DEFAULT 0,     -- Count of processed articles
  posts_generated INTEGER DEFAULT 0,        -- Count of successful generations
  error_message TEXT,                       -- Error details if failed
  started_at TIMESTAMP DEFAULT NOW(),
  completed_at TIMESTAMP WITH TIME ZONE,
  created_at TIMESTAMP DEFAULT NOW()
);
```

Table: `agent_activity`

Real-time activity log for streaming UI updates.

```sql
CREATE TABLE agent_activity (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  run_id UUID REFERENCES agent_runs(id),
  type TEXT NOT NULL,                       -- info/tool/decision/error/success
  message TEXT NOT NULL,                    -- Human-readable message
  article_title TEXT,                       -- Associated article (if any)
  tool_name TEXT,                           -- Tool used (if applicable)
  created_at TIMESTAMP DEFAULT NOW()
);
```

Table: `agent_queue`

Stores generated social media posts.

```
CREATE TABLE agent_queue (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  news_item_id UUID REFERENCES news_items(id),
  x_post TEXT,                     -- X/Twitter content
  instagram_caption TEXT,          -- Instagram content
  facebook_post TEXT,              -- Facebook content
  tier_used INTEGER,               -- Processing tier (1-4)
  tools_used TEXT[],               -- Array of tool names used
  created_at TIMESTAMP DEFAULT NOW()
);
```

---

4. Feeder System (Detailed)

4.1 RSS Feed Source

URL: `https://news.google.com/rss/search?q=pakistan&hl=en-PK&gl=PK&ceid=PK:en`

When freshness filter is set, URL becomes:

```
https://news.google.com/rss/search?q=pakistan+when:{X}h&hl=en-PK&gl=PK&ceid=PK:en
```

Where `{X}` is the freshness hours (1, 2, 6, 12, 24).

4.2 Deduplication Algorithm

```
function generateNewsHash(title: string, sourceName: string | null): string {
  // 1. Normalize title (lowercase, remove punctuation)
  const normalizedTitle = title.toLowerCase().replace(/[^\w\s]/g, '');

  // 2. Combine with source name
  const content = `${normalizedTitle}|${sourceName || 'unknown'}`;

  // 3. Generate SHA-256 hash
  return crypto.createHash('sha256').update(content).digest('hex');
}
```

4.3 Refresh Flow (Step-by-Step)

```
1. User clicks "Refresh" OR auto-refresh timer triggers
 ↓
2. DELETE all processed articles (is_posted = true)
 → Cleanup: "Deleted 45 processed articles"
 ↓
3. Fetch RSS feed with freshness filter
 → Request: GET google.com/rss/...?when:2h
 → Response: 106 articles from last 2 hours
 ↓
4. Filter by pub_date (double-check freshness)
 → Filtered: 98 articles within cutoff time
 ↓
5. Get existing hashes from database
 → SELECT hash FROM news_items
 ↓
6. Compare and filter duplicates
 → New: 23 unique articles
 → Duplicates: 75 already exist
 ↓
7. Insert new articles into database
 → INSERT 23 rows into news_items
 ↓
8. Enforce retention limit (max_retention setting)
 → If total > max: DELETE oldest articles
 → "Trimmed 15 old articles (retention: 100)"
 ↓
9. Update last_fetch timestamp in settings
 ↓
10. Return response to UI with counts
```

4.4 Settings Options

Setting	Values	Database Column	Behavior
Refresh Interval	5m, 10m, 15m, 30m, 1h	`refresh_interval` (ms)	Client-side setInterval timer
Auto-Refresh	On/Off	`is_active`	Enable/disable the timer
Max Retention	50, 100, 200, 300, 500	`max_retention`	Delete excess articles from oldest
Freshness	1h, 2h, 6h, 12h, 24h, All	`freshness_hours`	URL parameter + date filter

---

5. AI Agent System (Detailed)

5.1 Agent Architecture

```
┌─────────────────────────────────────────────────────────────┐
│                AGENT CONTROLLER                    │
│  lib/agent/index.ts                                  │
├─────────────────────────────────────────────────────────────┤
│                                             │
│  ┌─────────────┐   ┌─────────────┐   ┌─────────────┐     │
│  │   SYSTEM    │   │    TOOLS    │   │   OUTPUT    │     │
│  │   PROMPT    │   │ (LangChain) │   │   PARSER    │     │
│  └─────────────┘   └─────────────┘   └─────────────┘     │
│       │            │            │          │
│     ▼         ▼         ▼       │
│  ┌──────────────────────────────────────────┐  │
│  │           GROQ LLM              │  │
│  │  Models: gpt-oss-120b, llama-3.3-70b, llama-3.1-8b │  │
│  └──────────────────────────────────────────┘  │
│     │         │         │       │
│     ▼         ▼         ▼       │
│  ┌──────────┐  ┌──────────┐  ┌──────────┐   │
│  │ PINECONE │  │  JINA   │  │  SERPER  │   │
│  │   RAG   │  │ READER  │  │  SEARCH  │   │
│  └──────────┘  └──────────┘  └──────────┘   │
│                              │
└──────────────────────────────────────────────┘
```

5.2 Available Tools

Tool: `read_article`

Purpose: Scrape full article content from URL **API:** Jina AI Reader (https://r.jina.ai/) **Input:** `{ url: string }` **Output:** Article content in markdown format **When Used:** When snippet is insufficient for quality post

Tool: `search_web`

Purpose: Search Google for additional context **API:** Serper.dev Google Search **Input:** `{ query: string }` **Output:** Top 5 search results with snippets **When Used:** Need background info, verify facts, find related stories

Tool: `search_news`

Purpose: Search Google News for related stories **API:** Serper.dev News Search **Input:** `{ query: string }` **Output:** Recent news articles on topic **When Used:** Find multiple perspectives, verify breaking news

Tool:

queryKnowledge (RAG)

Purpose: Search past decisions and patterns **API:** Pinecone Vector Search **Input:** `{ query: string }` **Output:** Similar past decisions and their outcomes **When Used:** Learn from previous article processing

Tool:

storeExperience (RAG)

Purpose: Save decision for future learning **API:** Pinecone Upsert **Input:** `{ decision: object }`

Output: Confirmation of storage **When Used:** After every successful processing

5.3 Tier System (Processing Levels)

```
┌──────────────────────────────────────────────────────────┐
│                TIER DETERMINATION                         │
│                                                           │
│   Tier = f(number_of_tools_actually_used)                 │
│                                                           │
│  ┌────────────────────────────────────────────────────┐   │
│  │ TIER 1: Direct Generation (0 tools)            │   │
│  │ ──────────────────────────────────────────── │   │
│  │ • Uses only title + snippet from RSS           │   │
│  │ • Fastest processing (~2 seconds)              │   │
│  │ • Used when headline is self-explanatory       │   │
│  │ • Example: "PM visits UAE for bilateral talks" │   │
│  └────────────────────────────────────────────────────┘   │
│                    ↓                                      │
│  ┌────────────────────────────────────────────────────┐   │
│  │ TIER 2: Article Reading (1 tool)               │   │
│  │ ──────────────────────────────────────────── │   │
│  │ • Calls read_article tool to get full content  │   │
│  │ • Medium processing (~5 seconds)               │   │
│  │ • Used when more context needed                │   │
│  │ • Example: Complex political story             │   │
│  └────────────────────────────────────────────────────┘   │
│                    ↓                                      │
│  ┌────────────────────────────────────────────────────┐   │
│  │ TIER 3: Research Mode (2+ tools)               │   │
│  │ ──────────────────────────────────────────── │   │
│  │ • Calls read_article + search_web/search_news  │   │
│  │ • Slowest processing (~10+ seconds)            │   │
│   │  • Used for breaking news, verification needed      │   │
│   │  • Example: Major incident requiring verification   │   │
│   └────────────────────────────────────────────────────────────┘   │
│                    ↓                         │
│   ┌────────────────────────────────────────────────────────────┐   │
│   │  TIER 4: Fallback Generation (forced)             │   │
│   │  ─────────────────────────────────────────────────────   │   │
│   │  • Used when LLM fails or returns invalid response  │   │
│   │  • Generates post from title directly              │   │
│   │  • Guarantees no article is skipped                │   │
│   │  • Format: "Breaking: {title} - Stay informed..."   │   │
│   └────────────────────────────────────────────────────────────┘   │
│                                              │
└──────────────────────────────────────────────────────────────────────┘
```

5.4 Processing Flow (Per Article)

```
START: Article { title, link, snippet }
  │
  ▼
┌─────────────────────────────────────────────┐
│ 1. LOG: "Processing article 3/10"    │
│    → Stream to UI activity log       │
└─────────────────────────────────────────────┘
  │
  ▼
┌─────────────────────────────────────────────┐
│ 2. BUILD PROMPT                      │
│    → Include title, snippet, link    │
│    → Include system instructions     │
│    → Request JSON output format      │
└─────────────────────────────────────────────┘
  │
  ▼
┌─────────────────────────────────────────────┐
```

```
│ 3. SEND TO GROQ LLM                │
│    → Model: settings.model         │
│    → Temperature: 0.7              │
│    → Max tokens: 2048              │
│    → Enable tool calling           │
└─────────────────────────────────────┘
  │
  ▼
┌─────────────────────────────────────┐
│ 4. TOOL CALLING LOOP               │
│    ┌────────────────────────────┐   │
│    │ Does LLM want to use tool? │   │
│    └────────────────────────────┘   │
│         │ YES       │ NO         │
│         ▼           ▼           │
│    ┌───────────┐ ┌───────────┐     │
│    │ Execute   │ │ Continue  │     │
│    │ Tool      │ │ to output │     │
│    └───────────┘ └───────────┘     │
│         │                  │
│         ▼                  │
│    ┌──────────────────────────┐    │
│    │ LOG: "Tool: read_article" │    │
│    │ → Stream to UI           │    │
│    └──────────────────────────┘    │
│         │                  │
│         ▼                  │
│    ┌──────────────────────────┐    │
│    │ Append tool result to    │    │
│    │ conversation and repeat  │    │
│    └──────────────────────────┘    │
└─────────────────────────────────────┘
  │
  ▼
┌─────────────────────────────────────┐
```

```
│ 5. PARSE JSON RESPONSE              │
│   Expected format:                  │
│   {                                 │
│     "decision": "generate",         │
│     "x_post": "...",                │
│     "instagram_caption": "...",     │
│     "facebook_post": "...",         │
│     "reasoning": "..."              │
│   }                                 │
└─────────────────────────────────────┘
  │
  ├── SUCCESS ──────────────────────┐
  │                                 │
  ▼                                 ▼
┌─────────────────┐     ┌─────────────────┐
│ Parse OK        │     │ Parse FAILED    │
│                 │     │                 │
│ Check decision: │     │ TIER 4 Fallback:│
│ - "generate" ✓  │     │ Generate from   │
│ - "skip" → ✗    │     │ title directly  │
│   (force gen)   │     │                 │
└─────────────────┘     └─────────────────┘
  │                                 │
  └────────────┬────────────────────┘
               │
               ▼
┌─────────────────────────────────────┐
│ 6. CALCULATE TIER                   │
│   tier = toolCalls.length           │
│   0 tools → Tier 1                  │
│   1 tool  → Tier 2                  │
│   2+ tools → Tier 3                 │
│   Fallback → Tier 4                 │
└─────────────────────────────────────┘
  │
```

```
 ▼
┌─────────────────────────────────────┐
│ 7. SAVE TO DATABASE                 │
│   → INSERT into agent_queue         │
│   → UPDATE news_items.is_posted=true│
│   → UPDATE news_items.x_post, etc.  │
└─────────────────────────────────────┘
 │
 ▼
┌─────────────────────────────────────┐
│ 8. LOG SUCCESS                      │
│   → "Generated (Tier 2)"            │
│   → Update run stats                │
│   → Stream to UI                    │
└─────────────────────────────────────┘
 │
 ▼
END: Move to next article
```

5.5 System Prompt (Simplified)

```
You are an AI social media content creator for Pakistan news.
For EVERY article you receive, you MUST generate posts for all three platforms.
There is NO option to skip articles.
OUTPUT FORMAT (JSON):
{
 "decision": "generate",
 "x_post": "Max 280 chars, engaging, hashtags",
 "instagram_caption": "Longer, storytelling, emojis, 5-10 hashtags",
 "facebook_post": "Conversational, question to engage, link",
 "reasoning": "Brief explanation of your approach"
}
TOOLS AVAILABLE:
- read_article: Get full article content
- search_web: Search Google for context
```

- search_news: Search Google News

Use tools when needed, but always generate content.

5.6 Auto-Run Timer

```
// State
const [autoRunEnabled, setAutoRunEnabled] = useState(false);
const [autoRunInterval, setAutoRunInterval] = useState(60); // minutes
const [countdownSeconds, setCountdownSeconds] = useState(0);
const startAgentRef = useRef<() => void>(() => {});
// Timer Effect
useEffect(() => {
  if (autoRunEnabled && !isRunning) {
    setCountdownSeconds(autoRunInterval * 60);

    const timer = setInterval(() => {
      setCountdownSeconds(prev => {
        if (prev <= 1) {
          startAgentRef.current(); // Trigger agent run
          return autoRunInterval * 60; // Reset
        }
        return prev - 1;
      });
    }, 1000);

    return () => clearInterval(timer);
  }
}, [autoRunEnabled, autoRunInterval, isRunning]);
```

5.7 Preview Queue

Fetches upcoming articles based on current batch_size:

```
// API: GET /api/agent?preview=true
const upcomingArticles = await supabase
  .from('news_items')
  .select('id, title, source_name, pub_date')
  .eq('is_posted', false)
  .order('pub_date', { ascending: orderDirection === 'asc' })
  .limit(batch_size);
```

---

6. Posts System (Detailed)

6.1 Social Preview Cards

Each platform has a styled preview card:

Platform	Theme	Character Limit	Special Features
X/Twitter	Dark (#15202b)	280	Action buttons, char count
Instagram	Gradient (pink→orange)	2200	Hashtag styling, image placeholder
Facebook	Blue (#1877f2)	63,206	Link preview, engagement UI

6.2 Delete All Flow

```
1. User clicks "Delete All Posts"
 ↓
2. Confirmation modal appears
 → "Are you sure? This will delete X posts"
 ↓
3. User confirms
 ↓
4. API: DELETE /api/posts
 ↓
```

```
5. Backend actions:
 a. DELETE FROM agent_queue
 b. UPDATE news_items SET is_posted=false, x_post=null, ...
 ↓
6. UI refreshes with empty state
```

---

7. API Reference

GET /api/feeder

Purpose: Get news items and stats **Params:** `?refresh=true` to fetch from RSS **Response:**

```
{
 "success": true,
 "items": [...],
 "totalCount": 45,
 "newCount": 12,
 "duplicatesSkipped": 33,
 "processedDeleted": 20,
 "retentionTrimmed": 5
}
```

POST /api/feeder

Purpose: Update feeder settings **Body:**

```
{
 "refresh_interval": 900000,
 "is_active": true,
 "max_retention": 100,
 "freshness_hours": 2
}
```

GET /api/agent

Purpose: Get agent status and settings **Params:** `?preview=true` to include upcoming articles

Response:

```
{
 "success": true,
 "settings": {...},
 "runs": [...],
 "activeRun": {...},
 "isRunning": false,
 "activity": [...],
 "upcomingArticles": [...]
}
```

POST /api/agent

Purpose: Control agent **Actions:**

- `{ "action": "start" }` - Start processing
- `{ "action": "cancel" }` - Stop current run
- `{ "action": "update_settings", "settings": {...} }` - Update config

GET /api/posts

Purpose: Get generated posts **Response:**

```
{
 "success": true,
 "posts": [...],
 "totalCount": 150
}
```

DELETE /api/posts

Purpose: Delete all posts and reset articles **Response:**

```
{
 "success": true,
 "deleted": 150
}
```

---

8. File Structure

my-app/

```
├── app/
│   ├── layout.tsx          # Root layout with theme provider
│   ├── page.tsx            # Home redirect
│   ├── agent/
│   │   └── page.tsx        # Agent UI (892 lines)
│   ├── feeder/
│   │   ├── page.tsx        # Feeder UI
│   │   └── components/
│   │       ├── FeedList.tsx    # Article list
│   │       ├── FeedSettings.tsx # Settings bar
│   │       └── FetchHistory.tsx # Fetch log
│   ├── posts/
│   │   └── page.tsx        # Posts with social previews
│   └── api/
│       ├── agent/
│       │   └── route.ts    # Agent API
│       ├── feeder/
│       │   ├── route.ts    # Feeder API
│       │   └── settings/
│       │       └── route.ts  # Settings API
│       └── posts/
│           └── route.ts    # Posts API
├── lib/
│   ├── supabase.ts         # Supabase client
│   ├── types.ts            # Shared TypeScript types
│   ├── rss-parser.ts       # RSS fetching & parsing
│   ├── deduplication.ts    # Hash generation
│   ├── feed-store.ts       # Feeder DB operations
│   └── agent/
│       ├── index.ts        # Main agent logic (500+ lines)
│       ├── types.ts        # Agent types
│       ├── store.ts        # Agent DB operations
│       └── tools/
│           ├── langchain-tools.ts  # Tool definitions
│           └── serper-search.ts    # Serper API wrapper
```

```
├── components/
│   ├── theme-toggle.tsx       # Dark/light mode
│   └── ui/                    # Shadcn components
└── .env.local                 # Environment variables
```

---

9. Environment Variables

```
# Supabase Configuration
NEXT_PUBLIC_SUPABASE_URL=https://xxx.supabase.co
NEXT_PUBLIC_SUPABASE_ANON_KEY=eyJhbGciOiJIUzI1NiIs...
# Groq LLM
GROQ_API_KEY=gsk_xxx...
# Jina AI (Article Scraping)
JINA_API_KEY=jina_xxx...
# Serper (Google Search)
SERPER_API_KEY=xxx...
# Pinecone (RAG Memory)
PINECONE_API_KEY=pcsk_xxx...
PINECONE_INDEX_NAME=news-agent
```

---

10. Summary of All Features

Category	Feature	Status
Feeder	RSS fetching from Google News	✅ Done
	Deduplication via content hash	✅ Done
	Auto-refresh timer (5m-1h)	✅ Done

	Max retention limit (50-500)	✅ Done
	Freshness filter (1h-24h)	✅ Done
	Auto-delete processed articles	✅ Done
Agent	Multi-model support (3 models)	✅ Done
	Tool-based research (5 tools)	✅ Done
	Tiered processing (1-4)	✅ Done
	Real-time activity streaming	✅ Done
	Guaranteed generation (no skips)	✅ Done
	Auto-run timer (15m-4h)	✅ Done
	Preview queue before processing	✅ Done
	Batch size & order settings	✅ Done

Posts	Platform-specific generation	✅ Done
	Social preview cards (X, IG, FB)	✅ Done
	One-click copy	✅ Done
	Delete all with confirmation	✅ Done
RAG	Knowledge storage (Pinecone)	✅ Done
	Experience learning	✅ Done

-