Project - High Level Design

on

Autonomous Retail Researcher Agent

**Course Name: Agentic AI**

Institution Name: *Medicaps University – Datagami Skill Based Course*

|  |  |  |
| --- | --- | --- |
| **Sr no** | **Student Name** | **Enrolment Number** |
| 1 | MOHIT MEHARDE | EN22CS301609 |
| 2 | NANCY AGRAWAL | EN22CS301634 |
| 3 | NIHARIKA PANWAR | EN22CS301645 |
| 4 | NIKHIL GARG | EN22CS301646 |
| 5 | MAHIMA SAHU | EN22CS301570 |
| 6 | NAMAN MANGROLIYA | EN22CS301628 |

Group Name: **Group 10D6**

Project Number: **AAI-21**

Industry Mentor Name: **Mr. Suraj Nayak**

University Mentor Name: **Prof. Nishant Shrivastava**

Academic Year: **2025-2026**

**Table of Contents**

1. Introduction

1.1 Scope of the Document

1.2 Intended Audience

1.3 System Overview

2. System Design

2.1 Application Design

2.2 Process Flow

2.3 Information Flow

2.4 Components Design

2.5 Key Design Considerations

2.6 API Catalogue

3. Data Design

3.1 Data Model

3.2 Data Access Mechanism

3.3 Data Retention Policies

3.4 Data Migration

4. Interfaces

5. State and Session Management

6. Caching

7. Non-Functional Requirements

7.1 Security Aspects

7.2 Performance Aspects

8. References

**1. Introduction**

This document presents the High-Level Design (HLD) for the AI-Powered Retail Intelligence Platform. It describes the overall system architecture, core components, data flows, technology choices, and the key design decisions that underpin the platform.

The platform automates competitive intelligence gathering for retail businesses by scraping product data from Amazon India and Flipkart, persisting it in a cloud database (MongoDB Atlas), and running AI-powered analysis to deliver actionable insights through a Streamlit dashboard.

**1.1 Scope of the Document**

This document covers the following:

* End-to-end system architecture and its major tiers
* Component design for the scraping engine, AI analysis engine, database layer, and UI dashboard
* Data flow between all internal modules and external services
* API contracts for the key modules and third-party integrations
* Non-functional requirements: performance, security, caching, and session management

The following are explicitly out of scope for this document:

* User authentication and multi-tenant access control (single-user system in the current version)
* Payment processing or direct e-commerce transaction handling
* Detailed class/method-level design (covered in the separate LLD document: docs/LLD.md)

**1.2 Intended Audience**

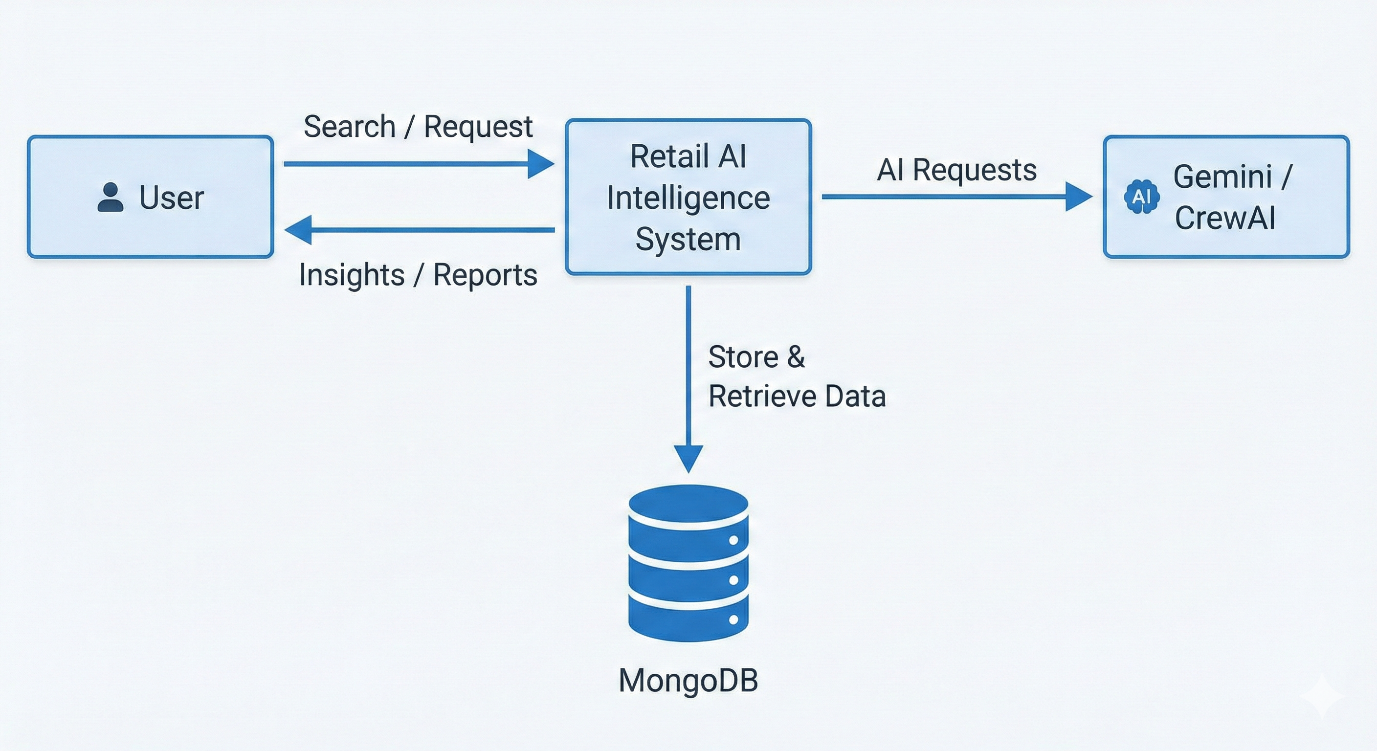
|  |  |
| --- | --- |
| **Audience** | **Purpose** |
| Project Developers / Engineers | Implementation reference and architectural guidance |
| University Mentors | Assessment and project evaluation |
| Industry Mentors | Technical review and feedback |
| Future Contributors | On-boarding and codebase understanding |

**1.3 System Overview**

The Retail AI Intelligence Platform is built in Python 3.11+ and composed of four principal layers:

|  |  |  |
| --- | --- | --- |
| **Layer** | **Technology** | **Responsibility** |
| Presentation | Streamlit 1.29+ | Interactive dashboard – KPIs, product explorer, price charts, AI insights, report archive |
| Application | Python 3.11+ | Scraping engine (Selenium + BeautifulSoup4), AI engine (Gemini single-agent + CrewAI multi-agent), PDF generator (ReportLab) |
| Data | MongoDB Atlas | Cloud NoSQL storage – products, price history, and generated reports |
| External Services | Gemini API, Groq API, SerpAPI | LLM inference for quick and deep market analysis; trend product search |

The system is currently a single-user local application intended to be scaled to a multi-user cloud deployment in Phase 3 of the roadmap.

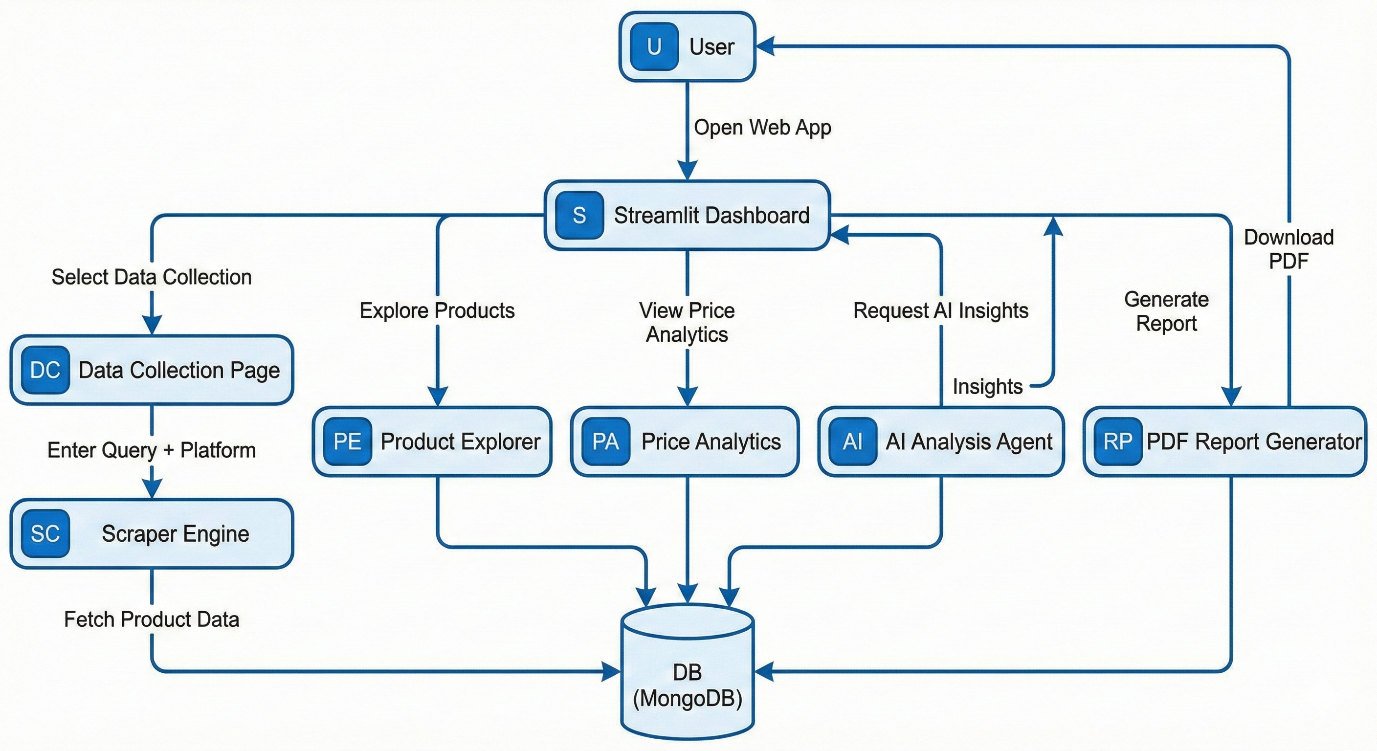


**2. System Design**

**2.1 Application Design**

The application follows a layered monolithic architecture for the current release. The three tiers interact as follows:

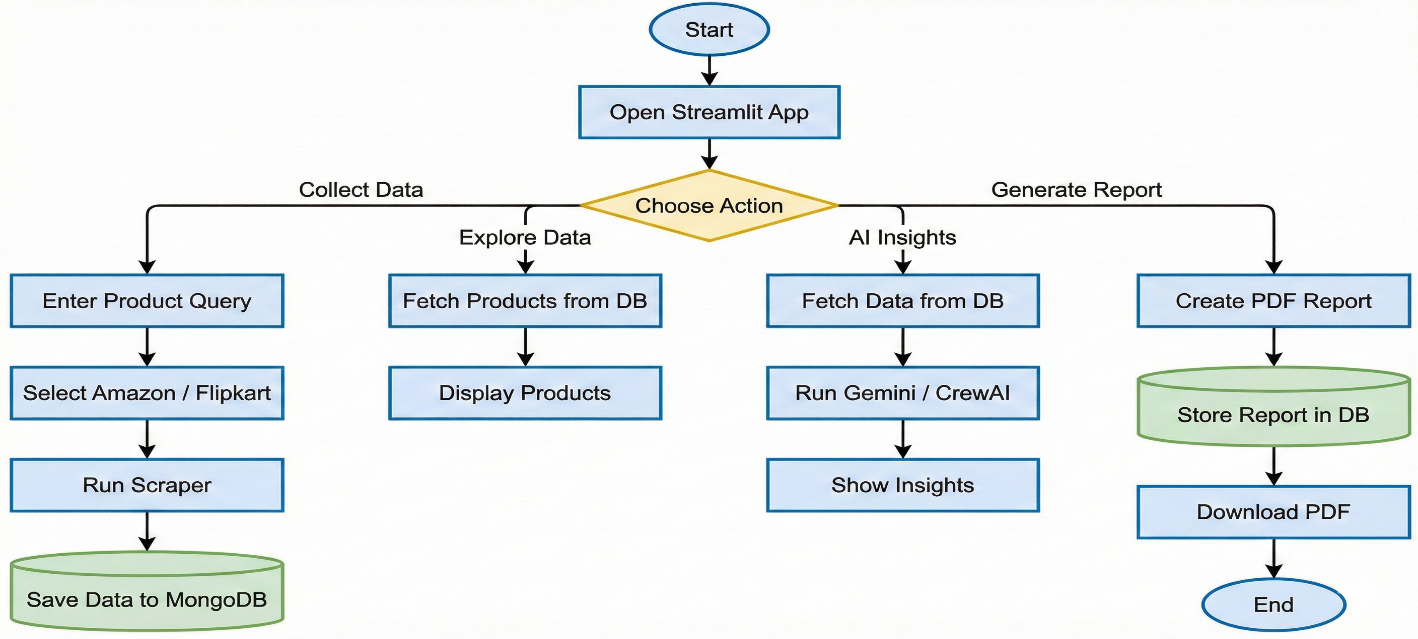
|  |  |  |
| --- | --- | --- |
| **Tier** | **Components** | **Interaction** |
| Presentation Tier | Streamlit Dashboard (dashboard.py) – 6 pages: Dashboard Home, Data Collection, Product Explorer, Price Analytics, AI Insights, Reports Archive | Browser calls Streamlit server; Streamlit calls Application tier directly (in-process function calls) |
| Application Tier | Scrapers (amazon\_scraper.py, flipkart\_scraper.py), AI Agents (analysis\_agent.py, crew\_manager.py), PDF Generator (pdf\_generator.py), Utilities (helpers.py) | Scrapers write to MongoDB via MongoManager; AI Agents read products from MongoDB; PDF Generator writes report binary back to MongoDB |
| Data Tier | MongoDB Atlas – three collections: products, price\_history, reports | PyMongo driver; connection string loaded from .env via Pydantic-Settings (config/settings.py) |



**2.2 Process Flow**

The end-to-end operational flow for a standard data collection and analysis cycle:

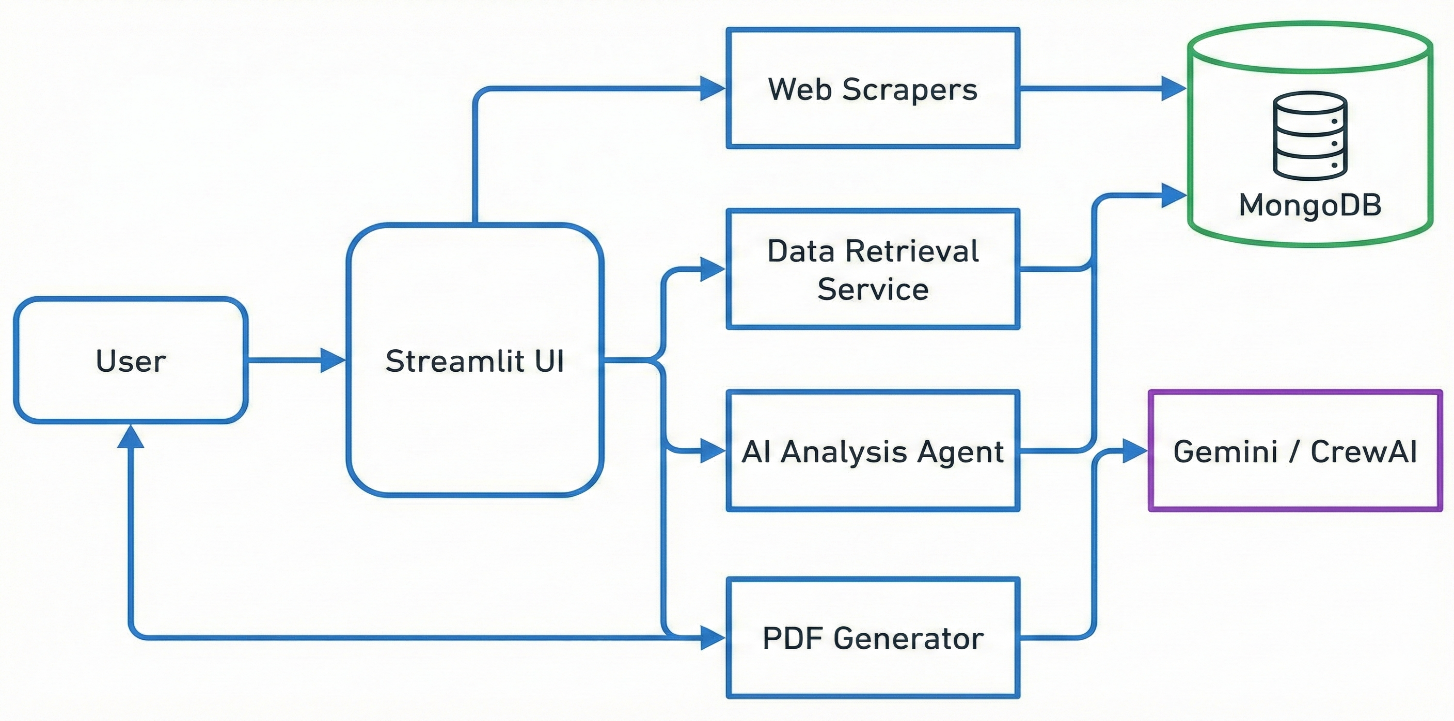
|  |  |  |  |
| --- | --- | --- | --- |
| **Step** | **Stage** | **Actor / Component** | **Output** |
| 1 | User Query Entry | User via Streamlit UI | Search term, platform, and category selections |
| 2 | Web Scraping | Amazon / Flipkart Scraper (Selenium + BS4) | List of raw product dicts (name, price, rating, reviews, URL, image) |
| 3 | Deduplication & Upsert | MongoManager.upsert\_product() | Products inserted or updated in products collection |
| 4 | Price History Record | MongoManager.add\_price\_history() | New price entry appended to price\_history collection |
| 5 | Quick Analysis (optional) | AnalysisAgent – Gemini 2.5 Flash | Markdown report: price range, top-rated, best value (~5–10 s) |
| 6 | Deep Analysis (optional) | CrewManager – CrewAI + Groq Llama 3.3 | Executive multi-section report via 5 specialised agents (~5–6 min) |
| 7 | PDF Generation | PDFGenerator – ReportLab | PDF binary saved to reports collection; download link shown in UI |
| 8 | Dashboard Render | Streamlit – dashboard.py | Charts, KPI cards, price tables, and download buttons rendered in browser |



**2.3 Information Flow**

The four primary information flows through the system are described below:

|  |  |  |  |
| --- | --- | --- | --- |
| **Flow** | **Source** | **Destination** | **Data Carried** |
| Inbound Scraping | Amazon.in / Flipkart.com | MongoDB Atlas – products | name, price, original\_price, rating, reviews\_count, url, image\_url, platform, category, scraped\_at |
| Price Tracking | MongoManager (on each scrape) | MongoDB Atlas – price\_history | product\_id, price, platform, recorded\_at |
| AI Analysis | MongoDB Atlas – products (JSON export) | Gemini API / Groq API → AI Agents | Product list as JSON (input); Markdown insights string (output) |
| Report Export | AI Analysis output (Markdown) | MongoDB Atlas – reports + browser download | PDF binary, report\_id, type, title, generated\_at |

****

**2.4 Components Design**

The application is organised into five packages inside the src/ directory. Each module has a single, well-defined responsibility:

|  |  |  |
| --- | --- | --- |
| **Package** | **Module** | **Responsibility** |
| src/scrapers/ | base\_scraper.py | Abstract base class: Selenium WebDriver lifecycle, retry logic (max 3x), configurable delay, user-agent rotation, explicit-wait helpers |
| src/scrapers/ | amazon\_scraper.py | Amazon India-specific CSS selectors, pagination handling, CAPTCHA detection and retry |
| src/scrapers/ | flipkart\_scraper.py | Flipkart-specific DOM selectors, category field mapping, price string normalisation |
| src/database/ | mongo\_manager.py | MongoDB Atlas connection pool management, all CRUD operations, upsert logic, price history append, report archive read/write |
| src/agents/ | analysis\_agent.py | Single-agent Gemini pipeline: builds structured system prompt with product JSON context, calls Gemini 2.5 Flash, returns Markdown string |
| src/agents/ | crew\_manager.py | CrewAI orchestration: defines 5 agents (Data Scout, Pricing Strategist, Risk Assessor, Demand Forecaster, Report Writer), 5 tasks, hierarchical process with Groq LLM |
| src/utils/ | pdf\_generator.py | ReportLab PDF construction: page layout, branded header/footer, tables, charts, section formatting |
| src/utils/ | helpers.py | Shared utilities: price string-to-float parser, SHA-256 product\_id generator, timestamp helpers, category normalisation |
| src/ui/ | dashboard.py | Streamlit page router: sidebar navigation, per-page render functions, Plotly/Altair chart builders, session-state management |
| config/ | settings.py | Pydantic-Settings model: typed .env loader for API keys, MongoDB URI, scrape\_delay, max\_retries; validated at application startup |

**2.5 Key Design Considerations**

**2.5.1 Scraping Resilience**

* Selenium's webdriver-manager automatically downloads and manages the correct ChromeDriver binary.
* A configurable scrape\_delay (default: 2 s) and max\_retries (default: 3) with exponential back-off reduce the risk of IP-blocking.
* CAPTCHA detection triggers an automatic pause-and-retry cycle before raising a failure to the UI.

**2.5.2 AI Agent Architecture**

* Quick Analysis uses a single synchronous Gemini 2.5 Flash API call, keeping latency under 10 seconds.
* Deep Analysis uses CrewAI's hierarchical multi-agent process: a manager agent assigns tasks to 5 specialist agents and collates their outputs into a final executive report.
* Groq's Llama 3.3 70B model is used for CrewAI to leverage its low-latency, high-throughput inference API at zero cost.

**2.5.3 Product Deduplication**

* Each product is assigned a unique product\_id by SHA-256-hashing the concatenation of platform name and product URL.
* MongoDB's update\_one with upsert=True prevents duplicate records on repeated scrapes of the same product.
* Price history is always appended (never updated) to maintain a full audit trail of price changes over time.

**2.5.4 Configuration and Secrets Management**

* All credentials (API keys, MongoDB URI) are loaded at startup via Pydantic-Settings from a .env file.
* A .gitignore rule ensures the .env file is never committed to the repository.
* Environment variable names are case-insensitive, simplifying deployment across Windows, macOS, and Linux.

**2.6 API Catalogue**

External APIs consumed by the platform:

|  |  |  |  |
| --- | --- | --- | --- |
| **API / Service** | **Provider** | **Usage in Platform** | **Auth Method** |
| Gemini 2.5 Flash | Google AI Studio | Quick single-agent market analysis (AnalysisAgent) | GEMINI\_API\_KEY (.env) |
| Llama 3.3 70B | Groq Cloud | Multi-agent deep analysis via CrewAI (CrewManager) | GROQ\_API\_KEY (.env) |
| MongoDB Atlas | MongoDB | All database read/write operations via PyMongo driver | MONGODB\_URI connection string (.env) |
| SerpAPI | SerpAPI | Trending product search queries (Phase 2 roadmap feature) | SERPAPI\_KEY (.env) |

Key internal module interfaces:

|  |  |  |  |
| --- | --- | --- | --- |
| **Caller** | **Callee** | **Interface / Method** | **Purpose** |
| dashboard.py | amazon\_scraper.py | AmazonScraper.scrape\_products(query, category) | Trigger Amazon data collection |
| dashboard.py | flipkart\_scraper.py | FlipkartScraper.scrape\_products(query, category) | Trigger Flipkart data collection |
| Scrapers | mongo\_manager.py | MongoManager.upsert\_product(product\_dict) | Persist or update a scraped product |
| Scrapers | mongo\_manager.py | MongoManager.add\_price\_history(product\_id, price) | Record a price observation |
| dashboard.py | analysis\_agent.py | AnalysisAgent.analyze(products\_json) | Run quick AI analysis |
| dashboard.py | crew\_manager.py | CrewManager.run\_analysis(products\_json) | Run deep multi-agent analysis |
| dashboard.py | pdf\_generator.py | PDFGenerator.generate(analysis\_text, title) | Build and return PDF binary |

**3. Data Design**

**3.1 Data Model**

MongoDB Atlas is used as the sole persistent data store. Three collections are defined, each with a clear document schema:

**Collection 1: products**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| \_id | ObjectId | Auto-generated MongoDB primary key |
| product\_id | String | SHA-256 hash of platform + URL. Used as the upsert key. Unique index. |
| name | String | Full product title as scraped |
| price | Float | Current selling price in INR |
| original\_price | Float | MRP or listed original price, if available |
| rating | Float | Star rating (0.0 – 5.0) |
| reviews\_count | Integer | Total number of customer reviews |
| url | String | Direct URL to the product page |
| image\_url | String | Product thumbnail image URL |
| platform | String | Source marketplace: 'amazon' or 'flipkart' |
| category | String | Normalised category label (Electronics, Clothing, etc.) |
| scraped\_at | DateTime (UTC) | Timestamp of the most recent scrape (updated on each upsert) |
| created\_at | DateTime (UTC) | Timestamp of the first insertion |

**Collection 2: price\_history**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| \_id | ObjectId | Auto-generated primary key |
| product\_id | String | Foreign reference to products.product\_id |
| price | Float | Price recorded at this point in time |
| platform | String | Platform at the time of recording |
| recorded\_at | DateTime (UTC) | UTC timestamp of this price observation |

**Collection 3: reports**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| \_id | ObjectId | Auto-generated primary key |
| report\_id | String | UUID v4 identifier for external reference |
| title | String | Human-readable report title |
| type | String | Analysis type: 'quick' or 'deep' |
| platform | String | Platform analysed |
| content | String | Raw Markdown output from the LLM agent |
| pdf\_bytes | Binary (BSON) | Serialised ReportLab PDF document |
| generated\_at | DateTime (UTC) | Report creation timestamp |

**3.2 Data Access Mechanism**

All database interactions are encapsulated in MongoManager (src/database/mongo\_manager.py). The following access patterns and indexes are used:

|  |  |  |
| --- | --- | --- |
| **Operation** | **PyMongo Method** | **Index** |
| Insert / update product | update\_one({product\_id}, $set, upsert=True) | product\_id – unique ascending |
| Record price observation | insert\_one(price\_record) | product\_id – ascending (for range queries) |
| List products by platform | find({platform: x}).sort(scraped\_at, -1) | platform – ascending |
| Fetch price history for a product | find({product\_id: x}).sort(recorded\_at, 1) | product\_id + recorded\_at – compound |
| Save generated report | insert\_one(report\_doc) | report\_id – unique ascending |
| List all reports | find({}).sort(generated\_at, -1) | generated\_at – descending |

The PyMongo driver maintains a connection pool (default maxPoolSize: 5) sufficient for the current single-user workload. The MongoDB URI is read from .env via Pydantic-Settings and never hard-coded.

**3.3 Data Retention Policies**

* products: records are retained indefinitely. The scraped\_at field is refreshed on every upsert cycle.
* price\_history: all records are append-only and retained without expiry to support long-term trend analysis.
* reports: retained in full (including PDF binary) until a user manually deletes a report from the Reports Archive page.
* Roadmap (Phase 2): a 180-day TTL index on price\_history.recorded\_at will be introduced to cap collection growth at scale.

**3.4 Data Migration**

No migration is required for the initial deployment as the application starts with an empty database. Future migration considerations are:

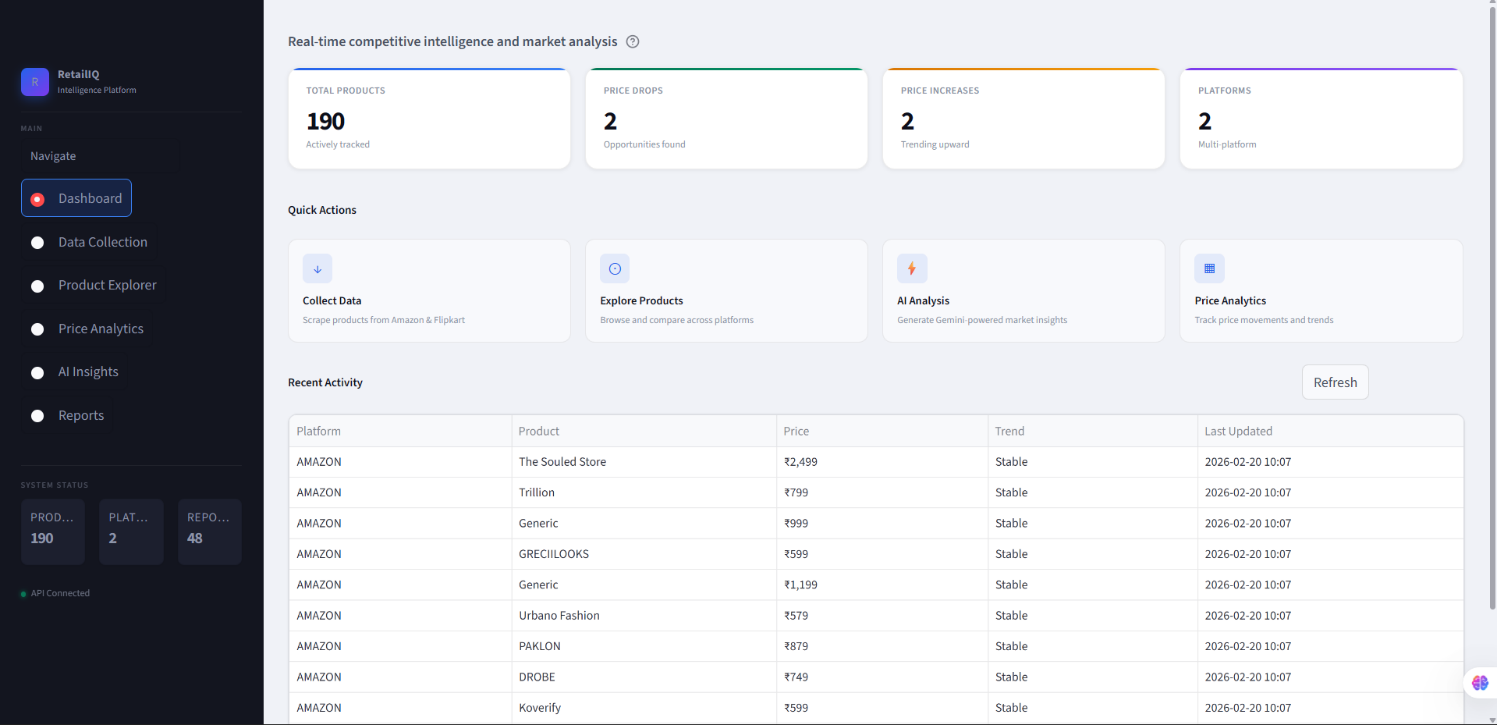
|  |  |
| --- | --- |
| **Scenario** | **Migration Approach** |
| Adding new required fields to the products schema | PyMongo update\_many script to back-fill default values for existing documents |
| Upgrading from MongoDB Atlas M0 to a paid tier | Standard mongodump / mongorestore tooling; no application code changes required |
| Schema versioning for breaking changes | Add a schema\_version integer field to each collection; migration scripts increment the version after transformation |

**4. Interfaces**

**4.1 User Interface**

The entire user interface is delivered through a Streamlit web application served at http://localhost:8501/. The dashboard has six pages, navigated via a persistent left-hand sidebar:

|  |  |
| --- | --- |
| **Page** | **Description** |
| Dashboard Home | High-level KPI cards (total products, platforms, recent scrapes, price drops). Quick-action buttons to jump to other pages. |
| Data Collection | Form inputs: search query string, platform dropdown (Amazon / Flipkart), category dropdown. Real-time progress indicator during scraping. Post-scrape summary. |
| Product Explorer | Filterable and sortable data table of all stored products. Clicking a row opens a detail panel with a price history line chart. |
| Price Analytics | Aggregate charts: price distribution histogram, top price-drop table, platform comparison bar chart, trend sparklines. |
| AI Insights | Toggle between Quick Analysis and Deep Analysis. Markdown-rendered output with section headings. Download-as-PDF button. |
| Reports Archive | Chronological list of previously generated reports with metadata (type, platform, date). Re-download PDF. Delete report. |



**4.2 External Service Interfaces**

|  |  |  |  |
| --- | --- | --- | --- |
| **Service** | **Interface Type** | **Data Format** | **Error Handling Strategy** |
| Amazon.in | HTTP via Selenium (headless Chrome) | HTML DOM → Python dict | CAPTCHA detected → pause 5 s → retry; max 3 attempts before surfacing error to UI |
| Flipkart.com | HTTP via Selenium (headless Chrome) | HTML DOM → Python dict | Selector mismatch → log warning → skip product; scrape continues |
| Google Gemini API | HTTPS REST (google-genai SDK) | JSON request → Markdown string response | API error / quota exceeded → display error message in AI Insights page |
| Groq API (via CrewAI / litellm) | HTTPS REST | JSON request → JSON response | Timeout set to 120 s; on failure CrewAI surfaces a partial result or error message |
| MongoDB Atlas | TCP (PyMongo driver) | BSON documents | Connection failure → retry 3× with 2 s back-off; persistent failure shows banner in UI |

**5. State and Session Management**

The platform uses Streamlit's built-in session\_state object for in-memory, per-tab state, combined with MongoDB Atlas for all durable persistence:

|  |  |  |  |
| --- | --- | --- | --- |
| **State Variable** | **Storage** | **Scope** | **Purpose** |
| st.session\_state.scraped\_products | Streamlit session\_state | Per browser tab | Caches the product list returned by the most recent scrape to avoid re-querying the DB on every page re-run |
| st.session\_state.analysis\_result | Streamlit session\_state | Per browser tab | Holds the last AI analysis Markdown string so the output can be re-rendered without re-calling the LLM API |
| st.session\_state.selected\_product\_id | Streamlit session\_state | Per browser tab | Tracks the product currently selected in the Product Explorer for the detail / price-history panel |
| st.session\_state.active\_page | Streamlit session\_state | Per browser tab | Maintains the current sidebar page selection across Streamlit re-runs |
| All product and price data | MongoDB Atlas | Persistent / shared | Durable storage of all scraped products, price history records, and generated reports |

There is no server-side user authentication or cross-session state sharing in the current release. Each browser tab is treated as an independent isolated session. Multi-user session management via a FastAPI backend layer is planned for Phase 3.

**6. Caching**

The following caching strategies are applied across the different layers of the platform:

|  |  |  |  |
| --- | --- | --- | --- |
| **Layer** | **Strategy** | **TTL / Scope** | **Details** |
| Scraping | No caching (always live) | N/A | Every scrape fetches fresh data from the marketplace to ensure price accuracy. Results are written to MongoDB immediately. |
| Database reads | Streamlit @st.cache\_data | 60 seconds | Product list queries and price history queries are cached for 60 s to reduce repeated MongoDB round-trips during dashboard interactions. |
| AI Analysis output | Streamlit session\_state cache | Per session | LLM analysis result is stored in session\_state after the first API call. Re-displaying the result does not re-invoke the API. |
| PDF Reports | MongoDB Binary (persistent) | Indefinite | Generated PDFs are stored as BSON Binary in the reports collection and served directly on download request without regeneration. |
| Future – Redis (Phase 3) | Shared in-memory cache | Configurable TTL | When migrating to a multi-user cloud deployment, a Redis layer will replace Streamlit session\_state caching to support shared state across instances. |

**7. Non-Functional Requirements**

**7.1 Security Aspects**

|  |  |
| --- | --- |
| **Threat / Concern** | **Mitigation Applied** |
| Credential exposure in source code | All API keys and the MongoDB URI are loaded exclusively from a .env file via Pydantic-Settings. A .gitignore rule prevents .env from being committed to version control. |
| IP blocking during scraping | Configurable scrape\_delay (default: 2 s), user-agent rotation in BaseScraper, and max\_retries (3) with back-off reduce the scraping footprint and detection risk. |
| Insecure report enumeration | Reports are referenced externally by a UUID v4 report\_id rather than a sequential integer, making enumeration impractical. |
| Dependency vulnerabilities | The GitHub Actions CI/CD pipeline runs bandit (SAST) and flake8 (style) checks on every push. Dependency pinning via requirements-lock.txt prevents uncontrolled upgrades. |
| LLM prompt injection via product data | Product data is JSON-serialised and injected into the system prompt as a structured context block. Free-text user input is not passed directly into prompt templates. |

**7.2 Performance Aspects**

|  |  |  |
| --- | --- | --- |
| **Requirement** | **Target** | **Implementation Strategy** |
| Dashboard initial render | < 2 seconds | Streamlit @st.cache\_data (TTL: 60 s) on all DB read calls. Charts are rendered lazily on tab selection. |
| Scrape throughput | ~20 products / min per platform | Sequential scraping with 2 s delay. Parallel tab-based scraping is planned for Phase 2 to double throughput. |
| Quick Analysis (Gemini) | < 10 seconds end-to-end | Gemini 2.5 Flash model selected for speed. Product JSON payload capped at the top 50 records. |
| Deep Analysis (CrewAI + Groq) | < 6 minutes end-to-end | CrewAI executes tasks in parallel where agent dependencies allow. Groq provides low-latency inference for Llama 3.3 70B. |
| MongoDB write latency | < 200 ms per upsert | PyMongo write concern: majority. Atlas M0 SSD-backed storage. Connection pool maintained throughout the session. |
| PDF generation | < 5 seconds | Synchronous ReportLab generation. The binary is persisted in MongoDB so subsequent downloads require no regeneration. |

**8. References**

|  |  |
| --- | --- |
| **Reference** | **Location / URL** |
| Project GitHub Repository | https://github.com/Nikhilgarg0/retail-ai-intelligence |
| Low-Level Design Document (LLD) | docs/LLD.md (within the repository) |
| Project Architecture Document | docs/ARCHITECTURE.md (within the repository) |
| Streamlit Documentation | https://docs.streamlit.io |
| MongoDB Atlas Documentation | https://www.mongodb.com/docs/atlas/ |
| PyMongo Driver Documentation | https://pymongo.readthedocs.io |
| Google Gemini API (google-genai) | https://ai.google.dev/gemini-api/docs |
| CrewAI Framework Documentation | https://docs.crewai.com |
| Groq Cloud API Reference | https://console.groq.com/docs |
| Selenium WebDriver – Python | https://selenium-python.readthedocs.io |
| BeautifulSoup4 Documentation | https://www.crummy.com/software/BeautifulSoup/bs4/doc/ |
| ReportLab PDF Library User Guide | https://www.reportlab.com/docs/reportlab-userguide.pdf |
| Pydantic-Settings Documentation | https://docs.pydantic.dev/latest/concepts/pydantic\_settings/ |
| bandit – Python Security Linter | https://bandit.readthedocs.io |

*— End of High-Level Design Document —*