InsightDash

KPI Intelligence Platform

Owen

October 1, 2025

Contents

1 Introduction

InsightDash is an analytics dashboard that pulls data from SQL Server and turns it into interactive charts. It features AI-powered summaries to explain key trends directly on the screen, providing a ready-to-use command centre for business leaders.

2 System Architecture

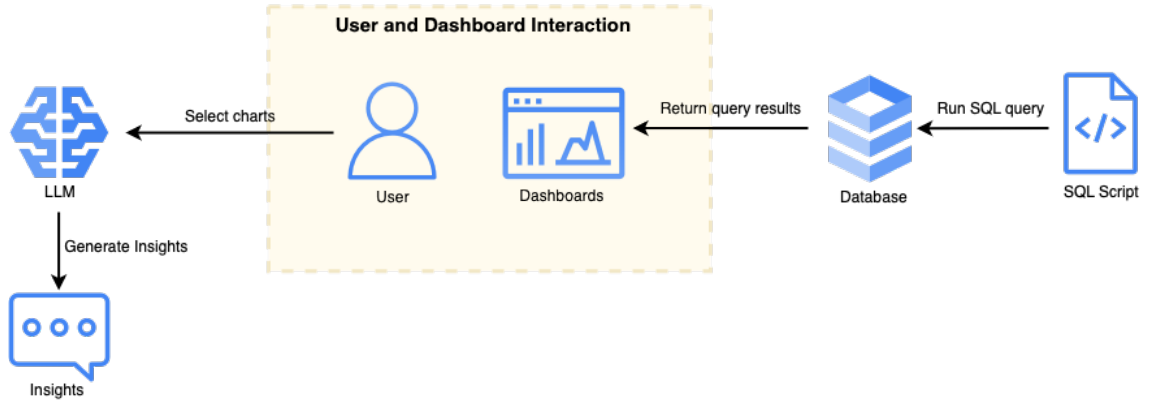
The runtime is orchestrated from the Dash entry point (`app.py`) and flows as follows:

1. Configure structured logging and read environment-driven settings (database, LLM, feature switches).
2. Fetch per-tab datasets by executing templated SQL through SQLAlchemy engines.
3. Normalise and remap query outputs to match the figure contracts expected by each tab.
4. Initialise Dash layouts, set up shared colour palettes, and register interactive callbacks within `app.py`.
5. Serve interactive charts that support cross-filtering, drill-down, and multi-select synchronisation.
6. Use the Gemini LLM to produce quantified narratives and insights for any graph on demand.
7. Persist usage logs and optional insight markdown while keeping the UI responsive.

2.1 Project Structure

```
InsightDash ..... Root for the interactive analytics app
├── app.py ..... Dash bootstrapper and callback wiring
├── app_tabs
│   ├── tab1 ..... Outlet performance view
│   │   ├── figures.py ..... Plotly graph builders
│   │   └── layout.py ..... Tab layout components
│   ├── tab2 ..... Dynamic scatter plot view
│   │   ├── figures.py
│   │   └── layout.py
│   └── tab3 ..... Trend and ranking view
│       ├── figures.py
│       └── layout.py
├── assets
│   ├── styles.css ..... Global styling overrides
│   └── sidebar_controls.css ..... Resizable panel theming
├── config
│   ├── settings.py ..... Environment flags and connection URIs
│   └── logging.py
├── data_layer
│   ├── base.py ..... Shared SQL execution helpers
│   ├── tab_1.py ..... Tab-specific query orchestration
│   ├── tab_2.py
│   └── tab_3.py
├── services
│   ├── llm.py ..... Gemini client abstraction
│   ├── insights.py ..... Chunked summarisation logic
│   └── prompts.py ..... Prompt builders for Gemini
├── sql_queries ..... Templated SQL per tab
├── utils
│   ├── data.py ..... Deduplication and packing helpers
│   ├── colors.py ..... Palette utilities and colour maps
│   ├── dataframe.py ..... NaN handling and concatenation helpers
│   └── df_summary.py ..... Basic statistics for LLM payloads
├── logs ..... Usage and diagnostic output
└── InsightDash.pdf ..... Project documentation (this file)
```

2.2 Component Diagram



1. **SQL Script**: Executes predefined queries.
2. **Database**: The data source queried to extract the necessary KPI data.
3. **Dashboards**: Visualizes the extracted data through interactive charts and graphs.
4. **Users**: Apply filters and request AI-generated insights.
5. **LLM**: Generates narrative insights based on the data from the user's selected charts.

3 Implementation

3.1 Runtime Algorithm

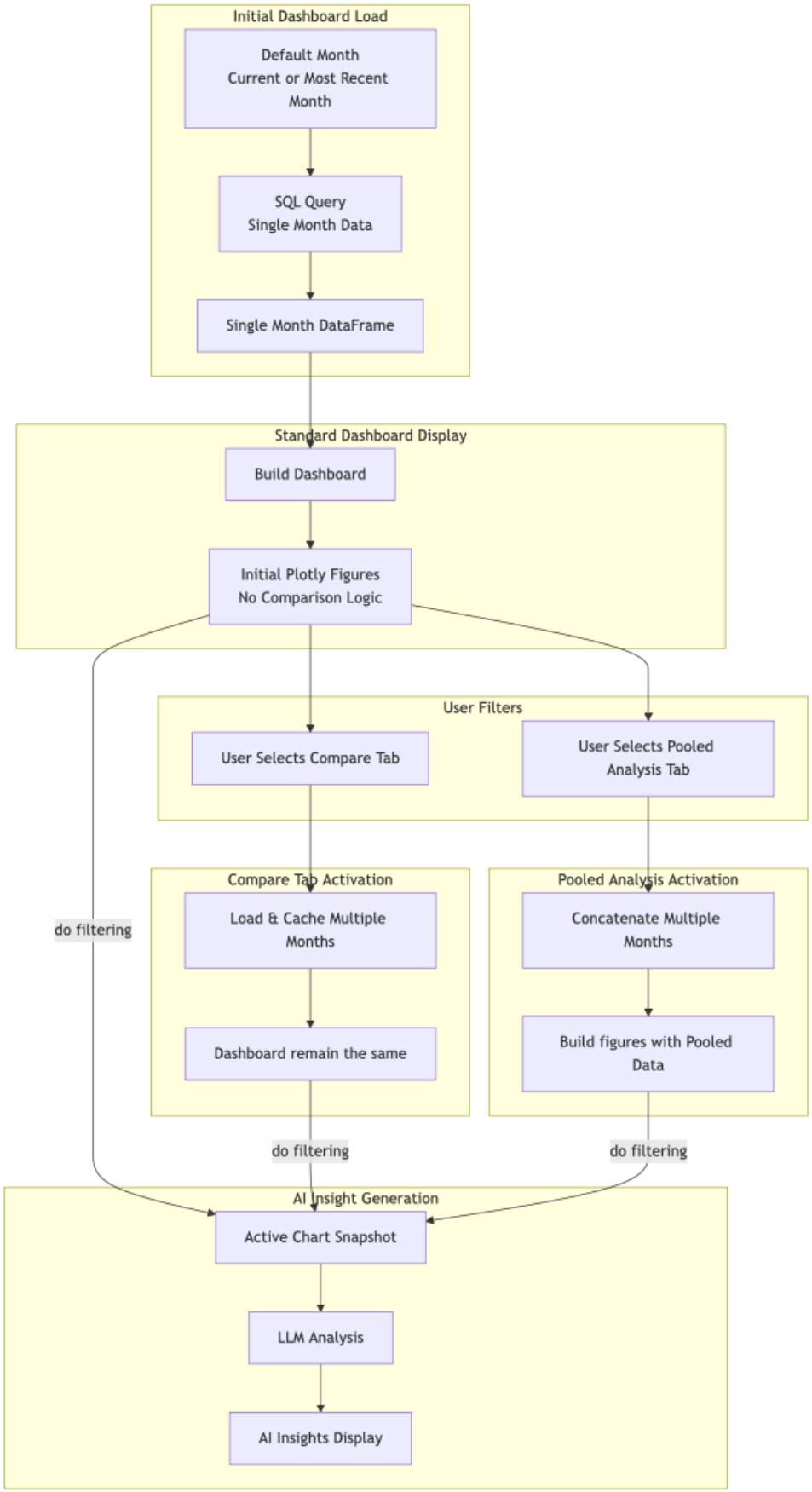
The Dash application follows the control flow summarised in Algorithm 1. Each stage is idempotent, enabling hot-reload in development and reliable deployments.

Algorithm 1 InsightDash Runtime Lifecycle (`app.py`)

Require: Environment variables, SQL templates, Plotly Dash runtime
Ensure: Responsive dashboard with optional LLM summaries

```
CONFIGURE_LOGGING
data_tab1 ← GET_TAB1_RESULTS("kpi_april")
data_tab2 ← GET_TAB2_RESULTS("kpi_april")
data_tab3 ← GET_TAB3_RESULTS("kpi_april")
monthly ← LOAD_ADDITIONAL_MONTHS(["kpi_may", ...])
app ← CREATE_DASHBOARD(data_tab1, data_tab2, data_tab3, monthly)
while app is running do
    Process callback inputs and recompute filtered DataFrames
    if user requests insight then
        slices ← CHUNK_DATAFRAME(selection)
        chunk_notes ← SUMMARIZE_CHART_VIA_CHUNKS(slices, context)
        markup ← SYNTHESIZE_ACROSS_CHARTS(chunk_notes)
        Update markdown panel with markup
    end if
    Append usage metrics to logs/usage.log
end while
```

3.2 Flow Chart



3.3 Key Modules

- `app.py`: Entry point that builds the Dash instance, injects tab layouts, and defines callback wiring for filters, graph selections, and insight generation.
- `data_layer/*`: Encapsulates SQL execution, result validation, and schema remediation so downstream charts always receive the expected columns.
- `app_tabs/*/figures.py`: Houses Plotly figure factories and table builders, isolating complex styling logic from callbacks.
- `services/insights.py`: Implements the map-reduce summarisation pipeline, including basic statistics to generate a consistent LLM outputs.
- `utils/colors.py` & `utils/data.py`: Provide consistent colour assignments, deduplication helpers, and DataFrame serialisation utilities shared across tabs.
- `assets/*.css`: Defines the visual identity (typography, spacing, panel controls) to keep charts and sidebars visually cohesive.

4 Related Technologies and References

- Plotly Dash documentation for component APIs and layout patterns.
- SQLAlchemy for engine creation and query execution against SQL Server.
- Google Gemini API for configuring the LLM insight features.
- Loguru for advanced logging patterns used by the project.

5 Frequently Asked Questions

- **How do monthly KPI tables map to dashboard tabs?**
  Each tab module builds a SQL map using the configured month (e.g., `kpi_april`). The keys (`q1`, `q2`, etc.) line up with figure factories that expect specific schemas.
- **What ensures colour consistency across tabs?**
  `utils.colors.color_map_from_list` generates deterministic palette assignments based on sorted category lists, and the results are cached per session.
- **Where are usage analytics stored?**
  Loguru writes structured entries to `logs/usage.log`, tagging events with `usage=True` so downstream tooling can aggregate adoption metrics.
- **How do I add a new KPI tab?**
  Scaffold a new folder under `app_tabs`, implement `layout.py` and `figures.py`, then wire new callbacks inside `app.py` (or a dedicated module) and extend `data_layer` to supply the required SQL results.