📘 PRD: Chat-Based Analytics MVP (Python + n8n + Deterministic SQL)

Version: MVP v1
Goal: Build an embeddable chat analytics engine that converts natural-language questions → validated intent → safe SQL → chart image → chat response.

---

1. System Overview

A SaaS embeds a small chat widget.
User asks:

> "Revenue from Hindustan Aeronautics for landing gear in the last 12 months."

System flow:

```
Chat Widget → FastAPI /query → NLP Intent Parser → Validation Layer
→ SQL Builder → SQLite DB → Query Result → QuickChart → Widget (image)
```

n8n is used only for **tenant config setup** (mapping DB, columns, metrics).

---

2. Tech Stack

Backend

- Python 3.10+
- FastAPI
- SQLite (dummy DB for testing)
- SQLAlchemy optional (not required)

- OpenAI API (future — mock parser for MVP)
- QuickChart (chart image URLs)

Frontend (later)

- JS SDK loader
- Simple chat widget (HTML/CSS + fetch())

Tools

- n8n: for config setup (DB credentials + schema mapping)

---

3. Modules Required

```
/backend
   main.py                (FastAPI server)
   /nlp/intent_parser.py  (mock NL→intent function)
   /validation/validator.py
   /builder/sql_templates.py
   /builder/sql_builder.py
   /connector/db.py
   /chart/chart.py
   /config_store/tenant.json
/create_dummy_db.py
```

---

4. Dummy Database Requirements

Codex must generate a script called:

`create_enhanced_dummy_db.py`

Specs:

- DB: SQLite file `enhanced_sales.db`
- Table: `sales`
- Rows: 5,000
- Columns:
    - id (PK)
    - customer_name (TEXT) — 20 values
    - product_name (TEXT) — 10 values
    - region (TEXT) — 5 values
    - order_status (TEXT) — Open/Closed/Returned
    - sale_date (DATE) — random within last 3 years
    - unit_price (FLOAT) — realistic distribution
    - quantity (INT)

This is the dataset used for query testing.

---

5. Semantic Model Config

Store in JSON under `/config_store/tenant1.json`.

Example:

```
{
  "fact_table": "sales",
  "date_column": "sale_date",
  "metrics": {
    "revenue": "unit_price * quantity"
```

```
  },
  "dimensions": {
    "customer_name": "TEXT",
    "product_name": "TEXT",
    "region": "TEXT",
    "order_status": "TEXT"
  },
  "allowed_operations": ["sum", "count", "avg"],
  "date_ranges": ["last_12_months", "last_6_months", "last_3_months"]
}
```

---

6. Intent Schema

Intent parser must return **structured JSON**, not SQL.

```
{
  "metric": "revenue",
  "filters": {
    "customer_name": "Hindustan Aeronautics",
    "product_name": "Landing Gear"
  },
  "group_by": "month",
  "date_range": "last_12_months"
}
```

For MVP, Codex should build a **mock parser** based on simple string matching.

---

7. Validation Rules

Validator must check:

Metric validation

- `intent.metric` must exist in config.metrics

Filter validation

- each filter field must exist in config.dimensions

Date validation

- supported range only:
    - last_12_months
    - last_6_months
    - last_3_months

If invalid → throw safe error message.

---

8. SQL Templates (Deterministic)

Store in `/builder/sql_templates.py`

T1: Summary

```
SELECT SUM($metric_formula) AS value
FROM $fact_table
WHERE 1=1
  $filters
  AND $date_column BETWEEN '$start_date' AND '$end_date';
```

T2: Trend (default for MVP)

```
SELECT
    strftime('%Y-%m', $date_column) AS period,
    SUM($metric_formula) AS value
```

```
FROM $fact_table
WHERE 1=1
  $filters
  AND $date_column BETWEEN '$start_date' AND '$end_date'
GROUP BY 1
ORDER BY 1 ASC;
```

T3: Group-By

```
SELECT
    $group_by AS label,
    SUM($metric_formula) AS value
FROM $fact_table
WHERE 1=1
  $filters
  AND $date_column BETWEEN '$start_date' AND '$end_date'
GROUP BY 1
ORDER BY value DESC;
```

---

9. SQL Builder Requirements

File: `/builder/sql_builder.py`

Responsibilities:

- Select appropriate template (trend is default)
- Replace placeholders:
    - `$metric_formula`
    - `$fact_table`
    - `$date_column`

    - `$filters`
    - `$start_date`, `$end_date`

Filters must be rendered as:

```
AND customer_name = 'Hindustan Aeronautics'
AND product_name = 'Landing Gear'
```

Use `string.Template`.

---

10. Resolver Logic

File: `/validation/date_resolver.py`

For MVP:

```
last_12_months = today - 365 days
last_6_months  = today - 182 days
last_3_months  = today - 90 days
```

Return (start_date, end_date).

---

11. DB Connector

File: `/connector/db.py`

- uses SQLite
- function: `run_query(sql: str) -> list`
- open connection, execute, return rows

---

12. Chart Generator

File: `/chart/chart.py`

Use QuickChart.io:

```
def generate_chart_url(labels, values):
    payload = {
        "type": "line",
        "data": {
            "labels": labels,
            "datasets": [{"data": values}]
        }
    }
    return "https://quickchart.io/chart?c=" + json.dumps(payload)
```

---

13. FastAPI Endpoint

File: `main.py`

```
POST /query
Body:
{
  "tenant": "tenant1",
  "question": "revenue for landing gear..."
}

Process:
1. Load config
2. Parse intent
3. Validate intent
4. Build SQL
5. Run SQL
6. Build chart payload
```

```
7. Return:
{
  "answer": "...",
  "chart_url": "..."
}
```

---

14. End-to-End Example

Input:

```
Revenue from customer Hindustan Aeronautics for landing gear last 12 months
```

Intent:

```
{
 "metric": "revenue",
 "filters": {
    "customer_name": "Hindustan Aeronautics",
    "product_name": "Landing Gear"
 },
 "group_by": "month",
 "date_range": "last_12_months"
}
```

SQL generated:

```
SELECT
    strftime('%Y-%m', sale_date) AS period,
    SUM(unit_price * quantity) AS value
FROM sales
WHERE 1=1
  AND customer_name = 'Hindustan Aeronautics'
  AND product_name = 'Landing Gear'
  AND sale_date BETWEEN '2024-03-05' AND '2025-03-05'
GROUP BY 1
```

```
ORDER BY 1 ASC;
```

Response:

```
{
 "answer": "Revenue trend for Hindustan Aeronautics (Landing Gear) for last 12 months.",
 "chart_url": "https://quickchart.io/chart?c=...."
}
```

---

15. Success Criteria

- Query returns correct SQL for multiple filters
- SQL executes on dummy DB
- Chart URL is generated
- Chat output contains text + image
- No hallucinated SQL
- Intent strictly validated against config

---

16. Deliverables for Codex

Codex must generate:

1. `create_enhanced_dummy_db.py`
2. `/backend/main.py` (FastAPI)
3. `/nlp/intent_parser.py`

4. `/validation/validator.py`
5. `/validation/date_resolver.py`
6. `/builder/sql_templates.py`
7. `/builder/sql_builder.py`
8. `/connector/db.py`
9. `/chart/chart.py`
10. `/config_store/tenant1.json`

Each file must work independently and together.