Technical Design Document - Tukey

AI-powered decision intelligence platform for POS-driven retail businesses

---

1. Problem Statement

Retail businesses collect large volumes of **Point-of-Sale (POS)** data every day, but:

- Data is scattered across systems
- Insights are retrospective, not actionable
- Business owners struggle to answer questions like:
  - *What should I stock next?*
  - *Which product will be profitable next season?*
  - *Which customer segment should I target?*

Tukey solves this by:

1. Ingesting POS sales data
2. Visualizing trends using **Tableau dashboards**
3. Applying **basic predictive analytics**
4. Converting analytics into **natural-language business decisions** using an LLM

---

2. MVP Scope (Hackathon-Focused)

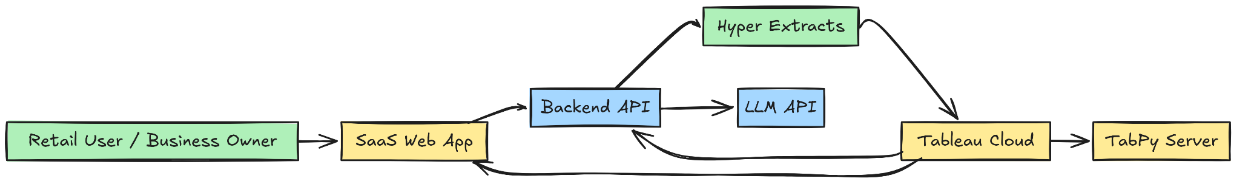
✅ Included in MVP

- Sample POS data ingestion
- Tableau-powered analytics dashboards
- Seasonal, product-wise, demographic insights
- Predictive trends (basic forecasting)

- Embedded Tableau dashboards in a SaaS UI
- LLM-based decision Q&A (rule + data driven)

---

3. High-Level Architecture



---

4. Technical Architecture and Design

System Architecture

A representative architecture for a Tableau-integrated web app is shown above. The frontend (e.g. React/JS) communicates with a Python-based backend (FastAPI/Flask), which in turn interfaces with Tableau (Cloud or Server) via REST APIs. In practice, FastAPI will handle authentication (e.g. OAuth2, JWT) and expose secured JSON/CSV endpoints [hoop.dev, tableau.com]. Tableau can consume these endpoints through its Web Data Connector or Embedding API. Data flows from the FastAPI backend to Tableau on demand or on schedule, avoiding manual exports [hoop.dev, hoop.dev]. For example, FastAPI can return only the sanitized, aggregated data that Tableau needs [hoop.dev]. Hosting is flexible: the FastAPI app can run in AWS (e.g., Docker on ECS/Fargate behind a load balancer, with RDS for persistence) [medium.com], while embedding dashboards from Tableau Cloud via Tableau's Embedding API or Connected Apps. Thus, the web app itself can be hosted on AWS or any cloud provider, independently of Tableau Cloud or Server: [tableau.github.io, medium.com].

Key components and technologies include:

- **Frontend:** JavaScript framework (e.g., React) that embeds Tableau visualizations using Tableau's Embedding API v3 or JS API.

- **Backend (FastAPI):** Serves as the API layer handling user requests, integrating with Tableau and the LLM. FastAPI is chosen for its high performance and ease of exposing LLM-powered endpoints: [datacamp.com, datacamp.com].

- **Tableau Integration:** Tableau REST API and Embedding API facilitate fetching workbooks/dashboards and embedding them in the app. Tableau *Connected Apps* can be configured to establish a trust (JWT or OAuth2) with the web app for secure, SSO-enabled embedding: [help.tableau.com, tableau.github.io].

- **Data Flow:** The frontend requests data or dashboards via the backend. The FastAPI backend authenticates the request, queries its database or Tableau APIs for data, and returns JSON to the frontend, which then renders or embeds the Tableau visualizations. This live pipeline ensures up-to-date dashboards without manual data dumps. [hoop.dev, hoop.dev].

- **LLM Integration:** A new "AI service" endpoint in FastAPI will relay user prompts to a large language model (e.g. GPT) and return generated insights. FastAPI's async capabilities make it well-suited for hosting such AI endpoints: [datacamp.com, datacamp.com].

By structuring the system this way, we ensure that *all* access is authenticated and authorized at the API edge, aligning Tableau's access with the app's RBAC, and allowing audit/logging on every data request: [hoop.dev, tableau.github.io].

Data Model (Entities)

The above entity-relationship diagram illustrates a typical **role-based access control (RBAC)** data model. Key entities include **User**, **Role**, and **Resource** (e.g., dashboards or data tables), plus linking tables **user_role** and **resource_role** to define many-to-many assignments and permissions. In this example, each *User* may have multiple *Roles* via `user_role`, and each *Resource* (e.g., "Sales Dashboard") is associated with roles and permissions (via `resource_role`). For instance, upon registration, the backend might assign the default "user" role to a new account [betterstack.com]. Initialization scripts create roles such as "user", "admin", "moderator" in the database [betterstack.com]. The FastAPI backend will use this model to enforce authorization: protected endpoints will declare required role scopes (e.,g. only "admin" can create new resources). Tableau content access can be tied to these roles so that embedded dashboards respect the same permissions [hoop.dev, tableau.github.io].

Additional domain entities include **Dashboard** or **Report** (metadata for Tableau views), **LLMQuery** (logging user prompts), and any business data tables needed for analytics. The Tableau integration uses the **Tableau REST API** to manage Tableau-side objects (users, projects, workbooks) in sync with our app's DB [tableau.github.io]. For example, when a new user is created, we could optionally use Tableau's API to provision a corresponding Tableau user or assign group attributes automatically. The `user_role` and `resource_role` tables shown above implement the core of our auth scheme, ensuring each API request can be checked against the user's roles and permissions (as recommended for secure embedding [hoop.dev]).

Authentication & Authorization Flows

FastAPI provides a robust security framework (OpenAPI-compliant) to handle auth. We will use OAuth2/JWT tokens for stateless auth, with password hashing on login. Upon successful login (via `/login`), FastAPI will issue a JWT containing the user's identity and assigned role scopes [betterstack.com, betterstack.com]. Each protected endpoint is decorated with dependencies that extract and verify the JWT, enforcing that the user's role scope matches the endpoint's requirements [betterstack.com, betterstack.com]. FastAPI's dependency-injection makes it easy to reuse authentication logic across routes [betterstack.com].

Additionally, for SSO or enterprise sign-on, Tableau **Connected Apps** can be configured: this establishes a trust (direct or OAuth2) between our app and Tableau Cloud/Server via JWT tokens [help.tableau.com, tableau.github.io]. For example, using OAuth2 trust, users log in through our identity provider (e.g. Okta), then our external app (FastAPI) can obtain a Tableau access token on the user's behalf [help.tableau.com]. This means a user authenticated in our app can seamlessly view embedded Tableau content without a second login. Direct-trust connected apps would allow our app to generate JWTs that Tableau accepts, enabling programmatic REST API calls and embedding without additional redirects [help.tableau.com, tableau.github.io].

Each user flow is audited: *authentication* confirms user identity, while *authorization* (RBAC) controls access to data and views [betterstack.com]. For example, a user with the "analyst" role might be allowed to view certain dashboards but not to edit them. These permissions are enforced in the FastAPI layer (using the Role-Resource model above) and consistently applied in Tableau via user filters or group membership [tableau.github.io].

Sequence of Operations

Below is an outline of key interaction flows in sequence form:

1. **User Login:** User submits credentials to FastAPI `/login`. The backend verifies the password (bcrypt) and, if valid, issues a signed JWT with the user's ID and roles. This token is returned to the frontend and stored in session/storage. Future requests include this JWT in the Authorization header. (*See [8] for FastAPI's auth framework: betterstack.com*.)

2. **Dashboard Request:** Authenticated user navigates to a page with an embedded Tableau dashboard. The frontend initializes the Tableau Embedding API, which may request an embed token from the backend if needed. FastAPI receives this (with JWT), validates the user, and then either redirects the Tableau JS to the Tableau Cloud URI or uses Tableau's REST API to generate a trusted ticket or VizQL session for embedding. The embedded view then loads in the user's browser.

3. **Data Fetch:** If the dashboard requires custom data not in Tableau extracts, the frontend (or the viz itself) may call a FastAPI data endpoint (e.g. `/api/data/sales?time=week`), including the JWT. The backend checks the user's roles and queries its own database or analytics model, returning JSON. Tableau (via a Web Data Connector) or the frontend JavaScript consumes this JSON to populate custom views or filters: [hoop.dev].

4. **LLM Query (MVP Feature):** The user enters a question or prompt in the UI. The frontend sends this text to FastAPI's `/api/ask-ai` endpoint with the JWT. FastAPI verifies the user and then forwards the prompt to an LLM service (e.g. OpenAI's API or a local GPT model). Upon receiving the LLM's response (with e.g. analysis or suggested queries), FastAPI returns it to the UI. This might then be displayed or used to filter the Tableau view. FastAPI makes this easy by exposing the LLM as a simple JSON REST API: [datacamp.com, datacamp.com].

Each of these flows is protected by FastAPI's security. For example, only users with the "admin" role can call certain management endpoints. The integration points with Tableau use secure connections and tokens, aligning with the best practices of mapping user scopes between the app and Tableau: [hoop.dev, tableau.github.io].

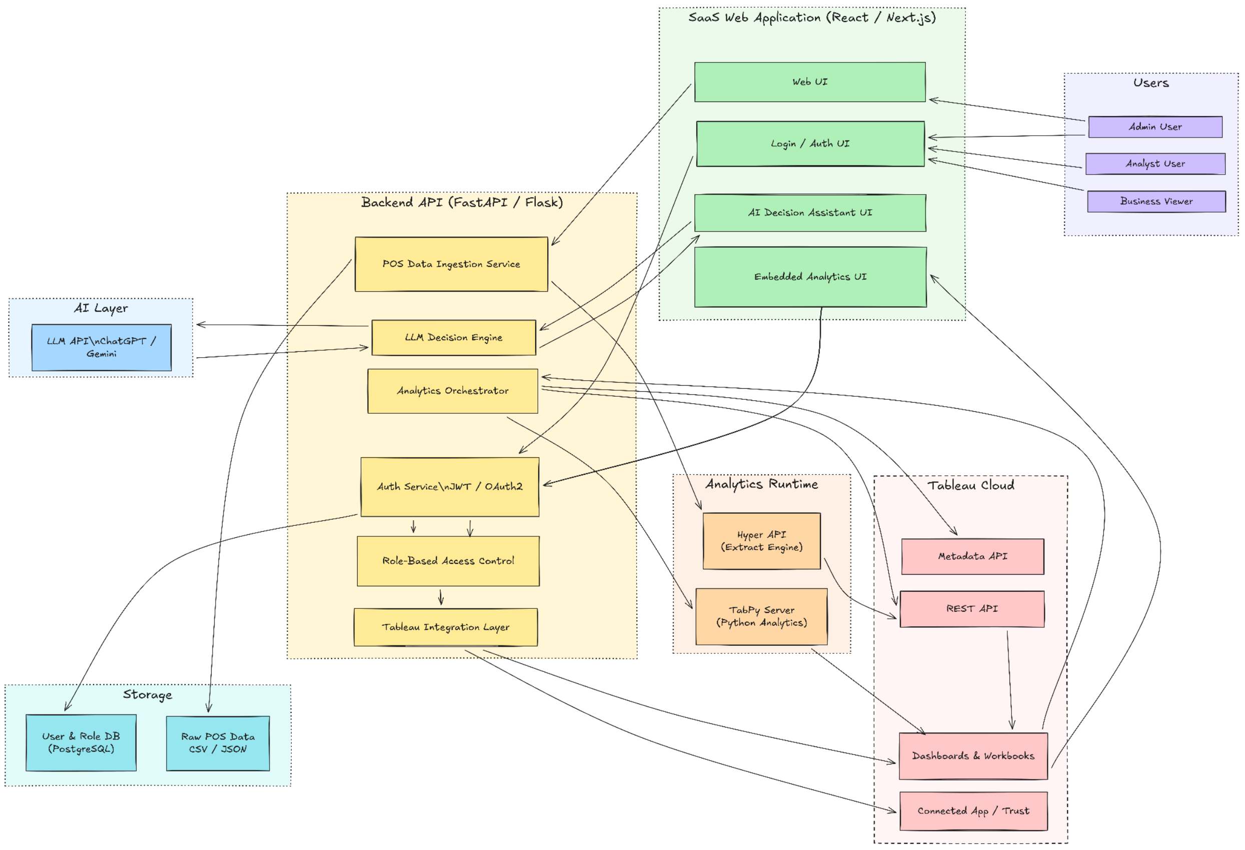
Deployment & Hosting

The system can be deployed on any cloud. FastAPI (or Flask) applications are commonly containerized (e.g., Docker) and run on AWS EC2/ECS, EKS, or serverless platforms: [medium.com]. For example, one might build a Docker image of the FastAPI service, push it to AWS ECR, and run it on ECS Fargate behind an Application Load Balancer (for HTTPS termination). A relational database (e.g. PostgreSQL on AWS RDS) can store user, role, and app data. Automated CI/CD pipelines (e.g. GitHub Actions) will handle builds and deployments.

Tableau Cloud (SaaS) manages its own hosting; our web app simply embeds from it. If instead we use Tableau Server (on-prem or on AWS), it can be hosted on AWS EC2 instances or VMs, since the REST API and VizQL service are self-contained. In either case, our frontend/backend remains a separate service. Communication between our AWS-hosted app and Tableau Cloud occurs over the public internet (using HTTPS and Tableau's APIs) – there is no requirement that they share a cloud provider. The Tableau REST API and Embedding API are the integration layers: [tableau.github.io].

---

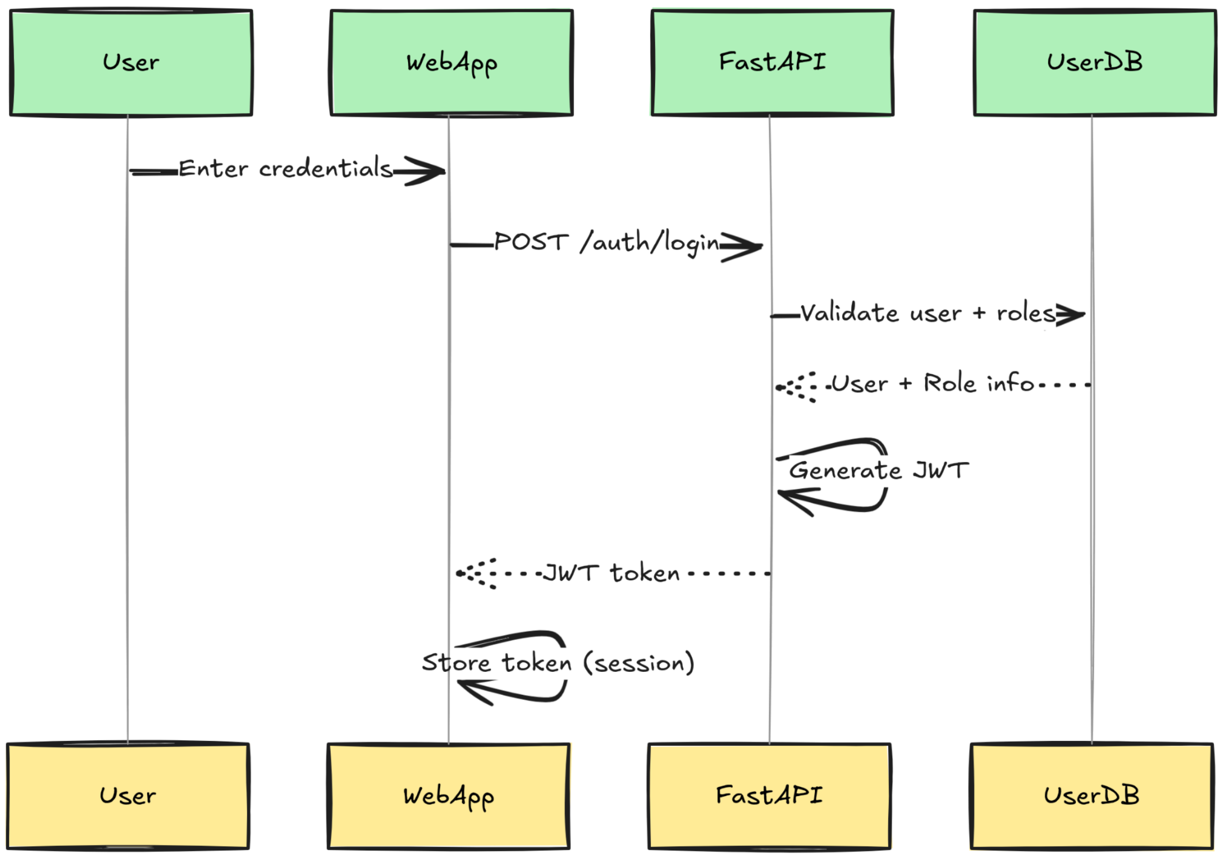
5. Complete System Architecture



---

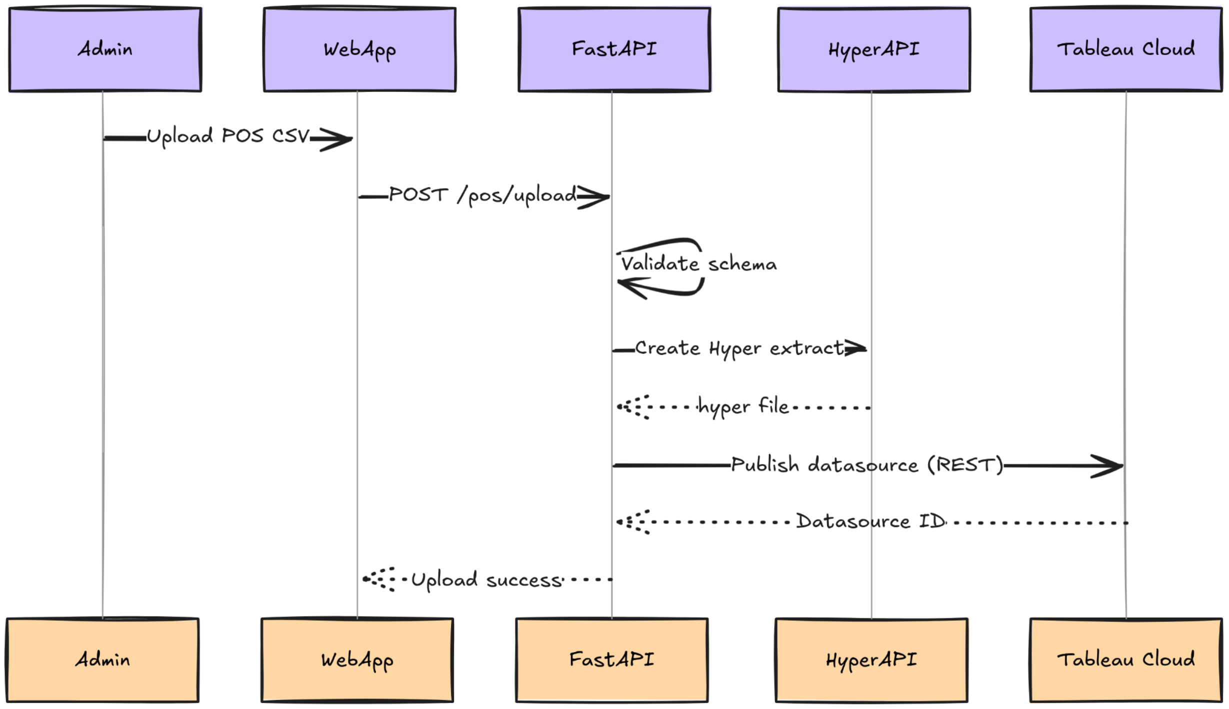
6. Sequence Diagrams

a. Authentication & Role-Based Access Flow



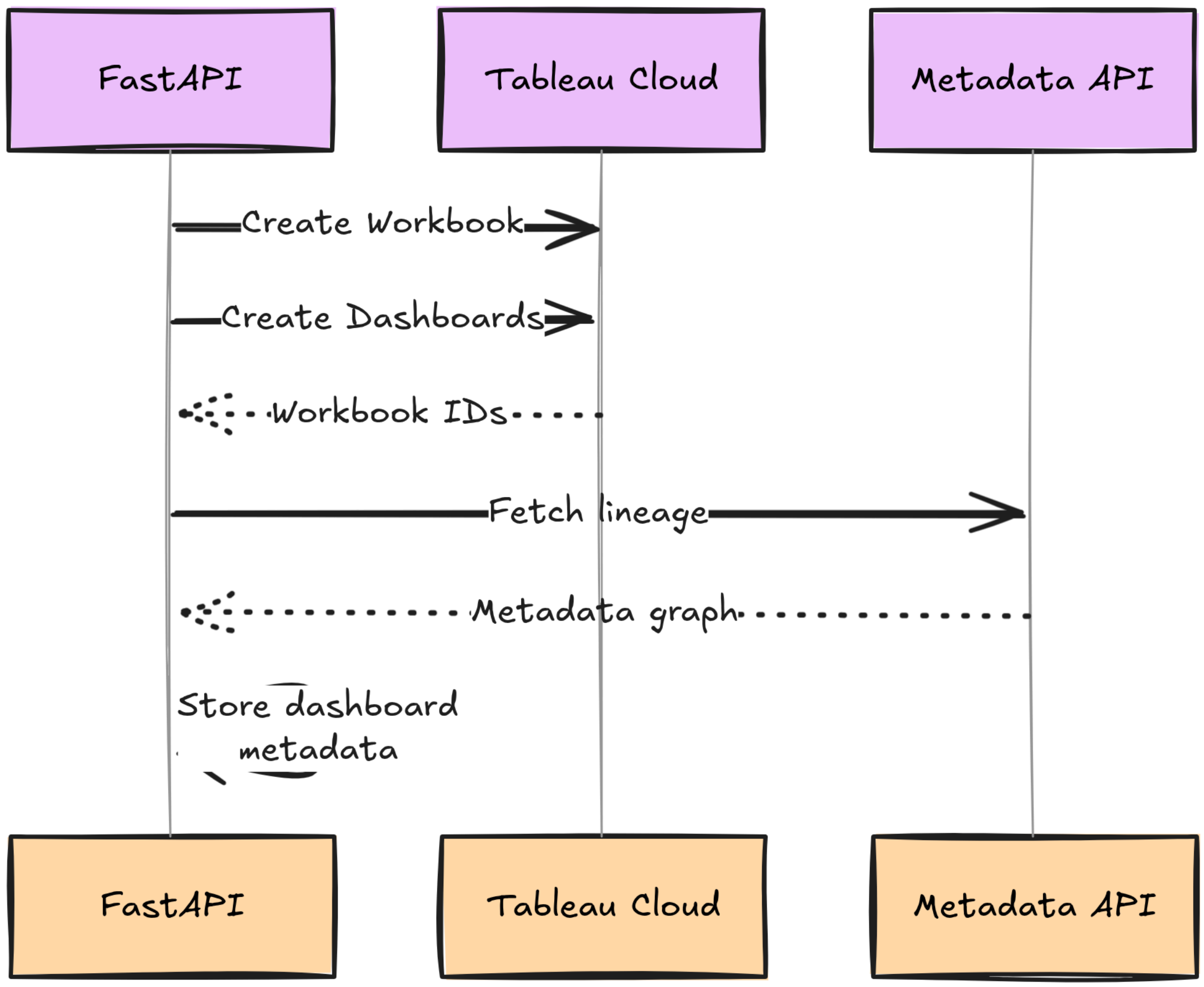
- Used by **all subsequent flows**
- JWT contains `user_id`, `org_id`, `roles`

b. POS Data Upload & Hyper Extract Creation



- MVP uses **sample data**
- Same flow supports **future incremental updates**

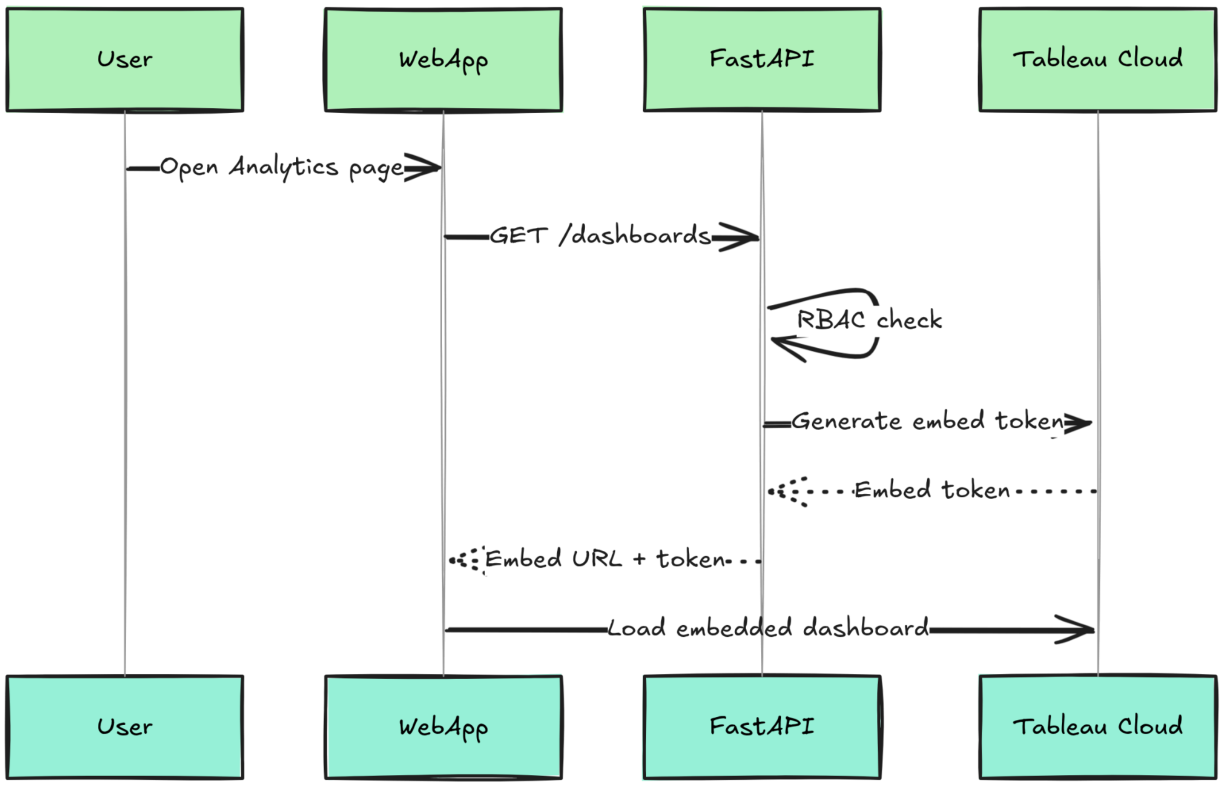
c. Dashboard Creation & Tableau Metadata Sync



Enables:

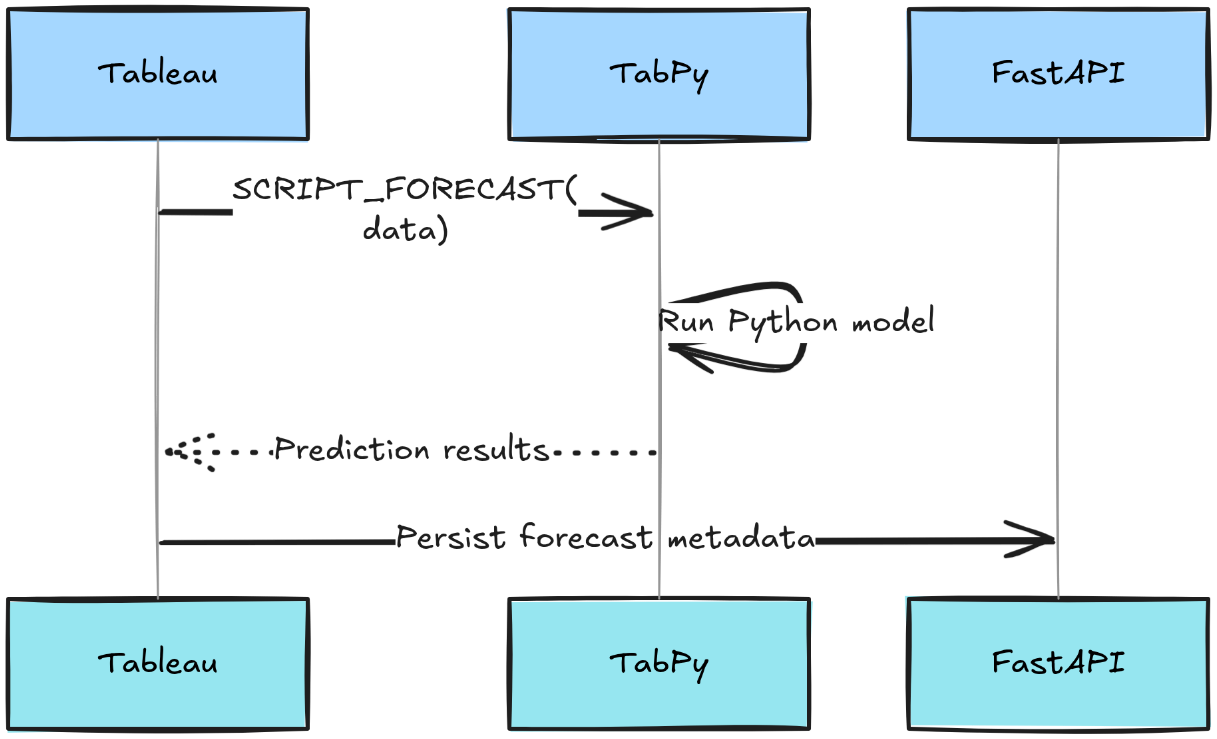
- Dashboard discovery
- RBAC → dashboard mapping
- Future lineage visualization

d. Embedded Dashboard Access (RBAC-Aware)



- Dashboards shown **only if permitted**
- Tableau Cloud is hosted independently

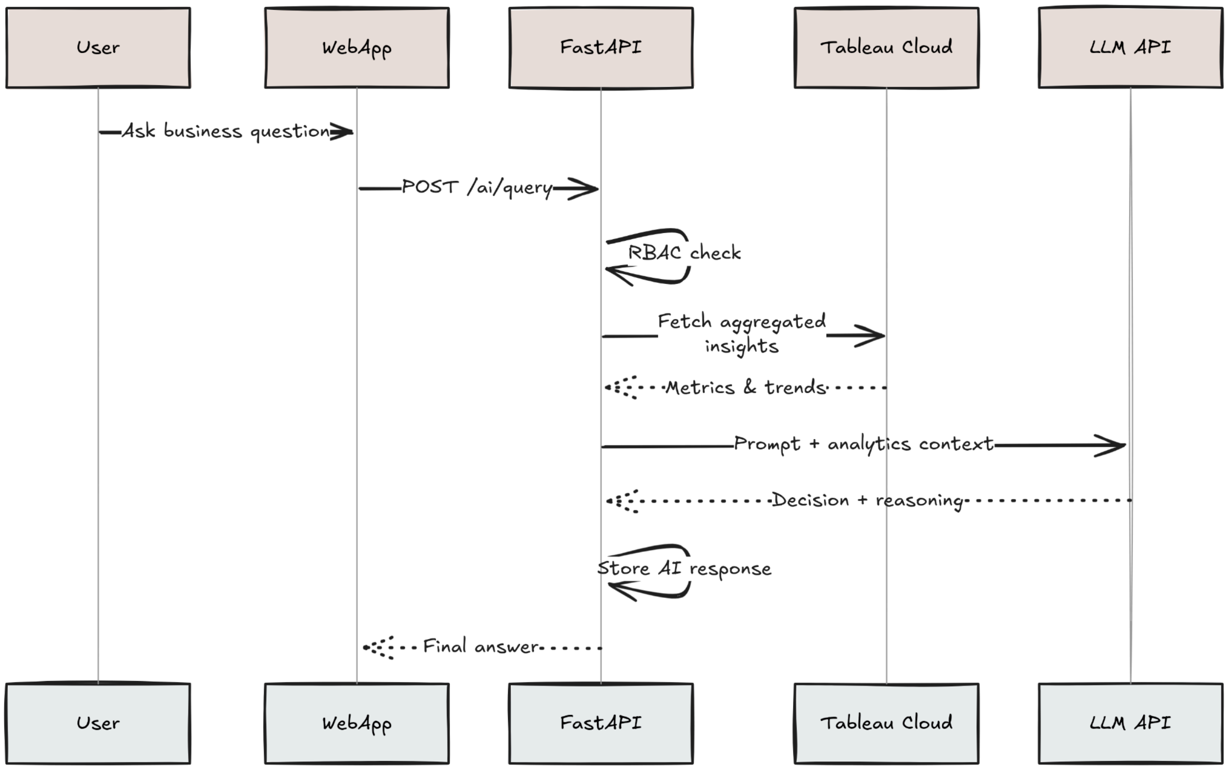
e. Predictive Analytics via TabPy



Supports:

- Seasonal forecasts
- Product demand predictions
- Revenue trend projections

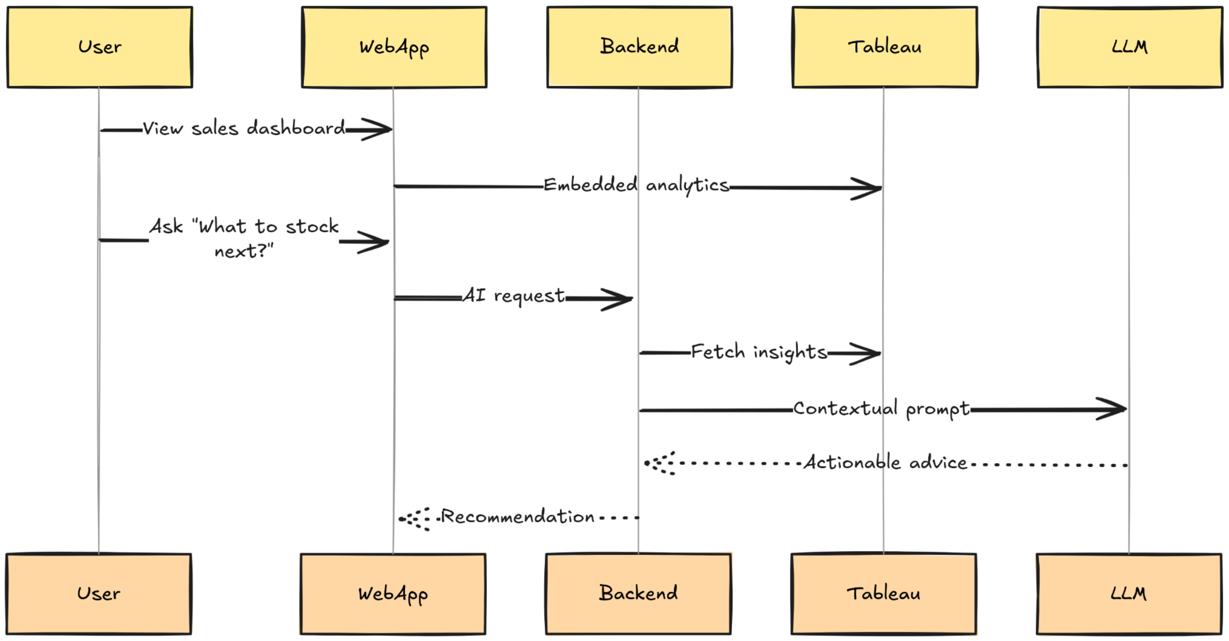
f. LLM-Based Business Decision Flow (Core MVP Feature)



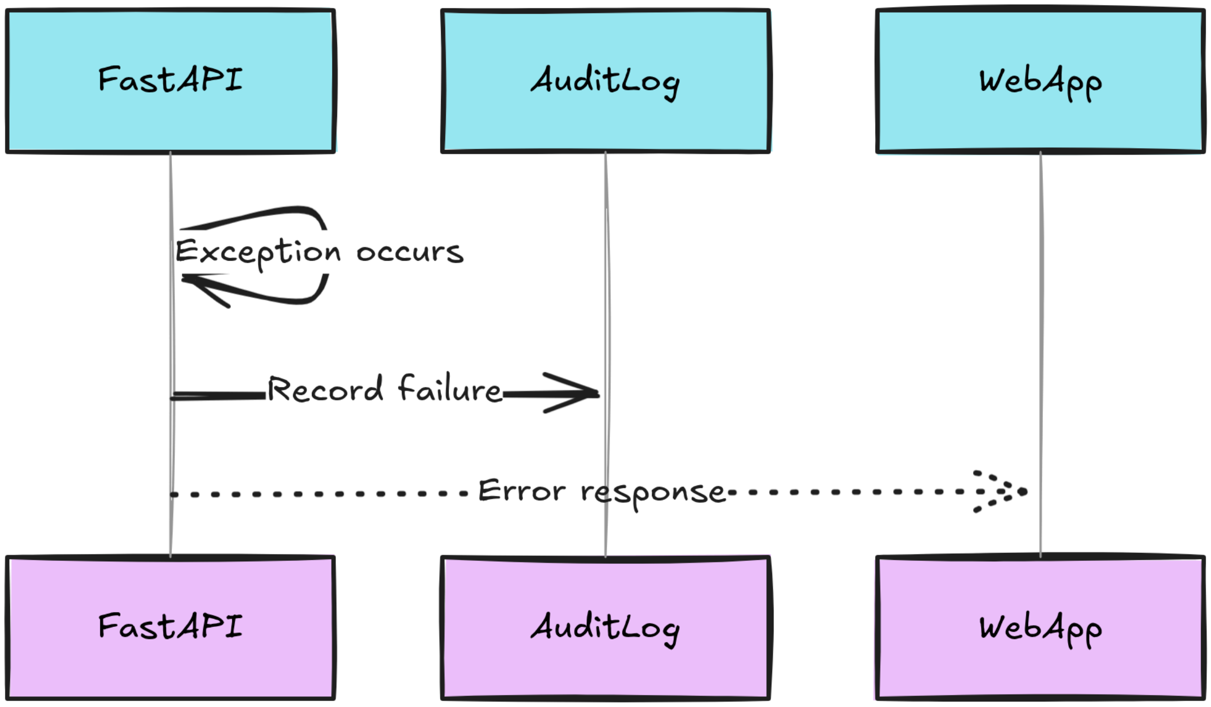
This is the **hackathon differentiator**

- Analytics → Context → **Decision**

g. End-to-End “Insight to Action” User Journey



h. Error Handling & Audit Logging



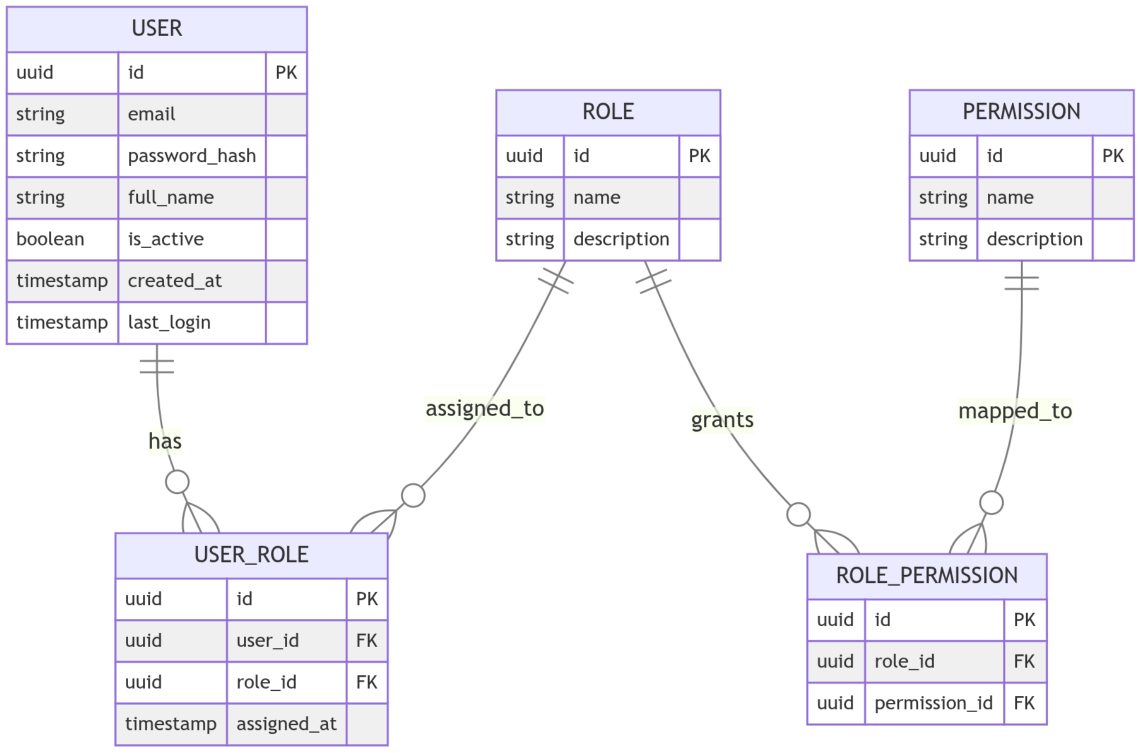
Ensures:

- Traceability
- Demo stability
- Debuggability

---

7. Entity Relationships

a. Core Identity & Access Management (IAM)



USER		
uuid	id	PK
string	email	
string	password_hash	
string	full_name	
boolean	is_active	
timestamp	created_at	
timestamp	last_login	

ROLE		
uuid	id	PK
string	name	
string	description	

PERMISSION		
uuid	id	PK
string	name	
string	description	

USER_ROLE		
uuid	id	PK
uuid	user_id	FK
uuid	role_id	FK
timestamp	assigned_at	

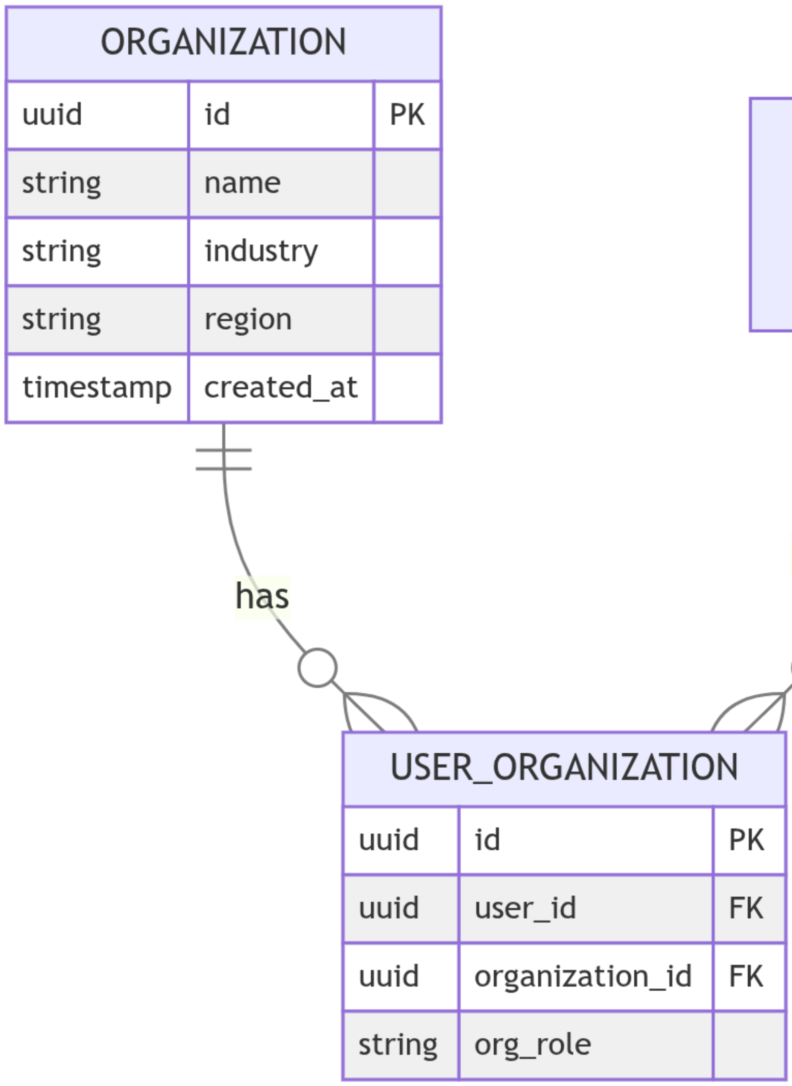
ROLE_PERMISSION		
uuid	id	PK
uuid	role_id	FK
uuid	permission_id	FK

Why this matters

- Enables **Admin / Analyst / Viewer**
- Enforced in FastAPI dependency layer
- Maps cleanly to Tableau groups later

---

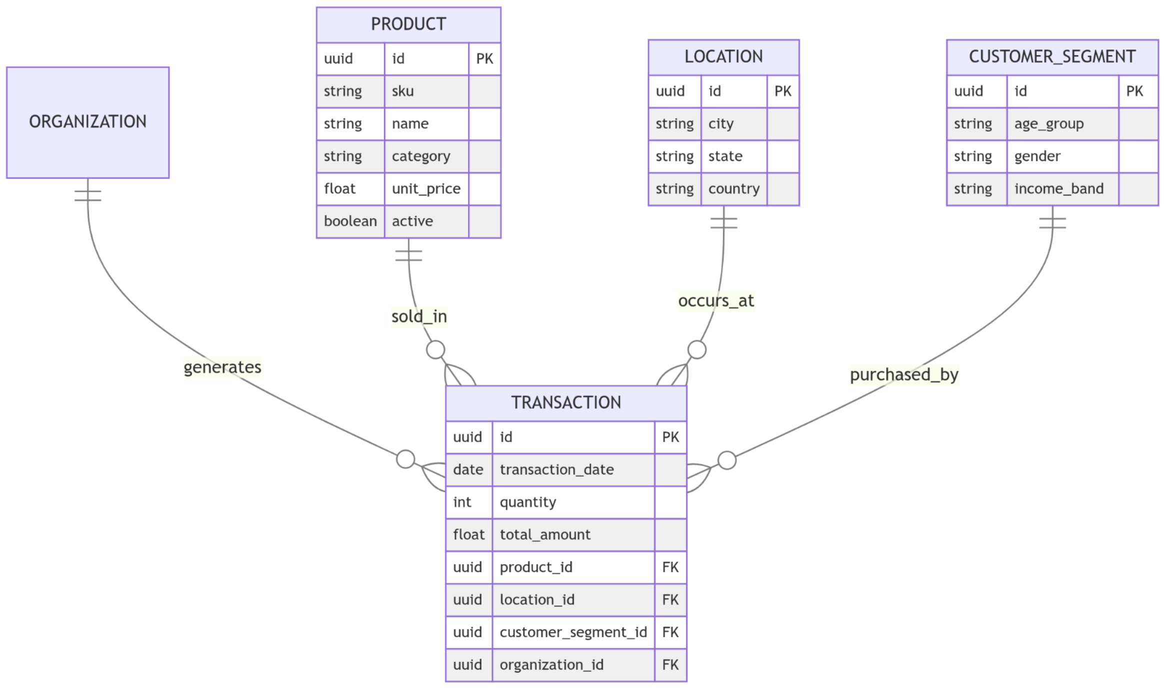
b. Organization / Tenant Model



MVP note:
You can start with a **single organization**, but this schema makes a multi-tenant future-proof.

---

c. POS & Business Data Model

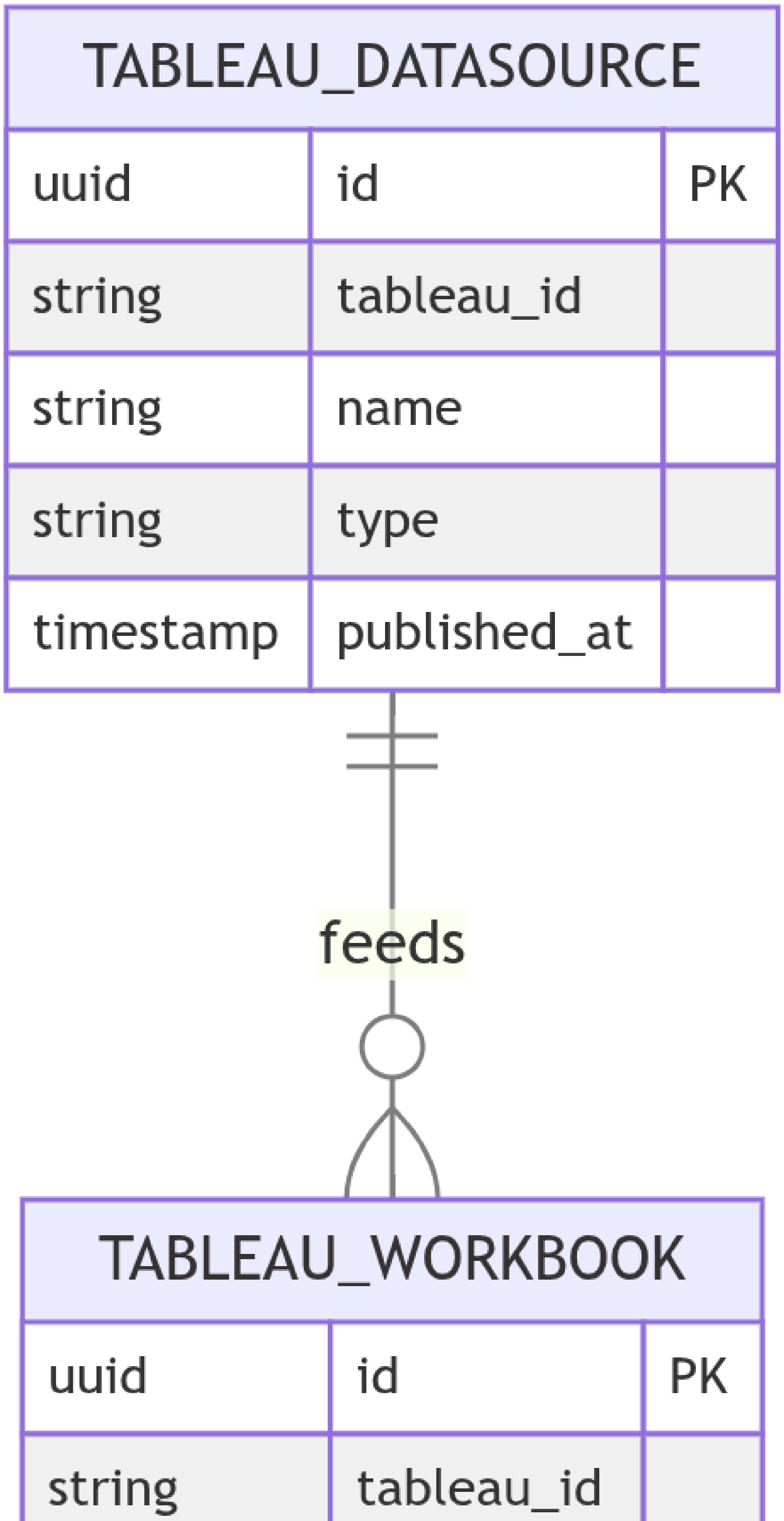


POS Data Characteristics

- **Fully compatible with:**
  - CSV
  - JSON
  - Hyper extracts
- **Optimized for:**
  - Time-series analytics
  - Seasonality
  - Demographics

---

d. Tableau Analytics & Metadata Layer



TABLEAU_DATASOURCE		
uuid	id	PK
string	tableau_id	
string	name	
string	type	
timestamp	published_at	

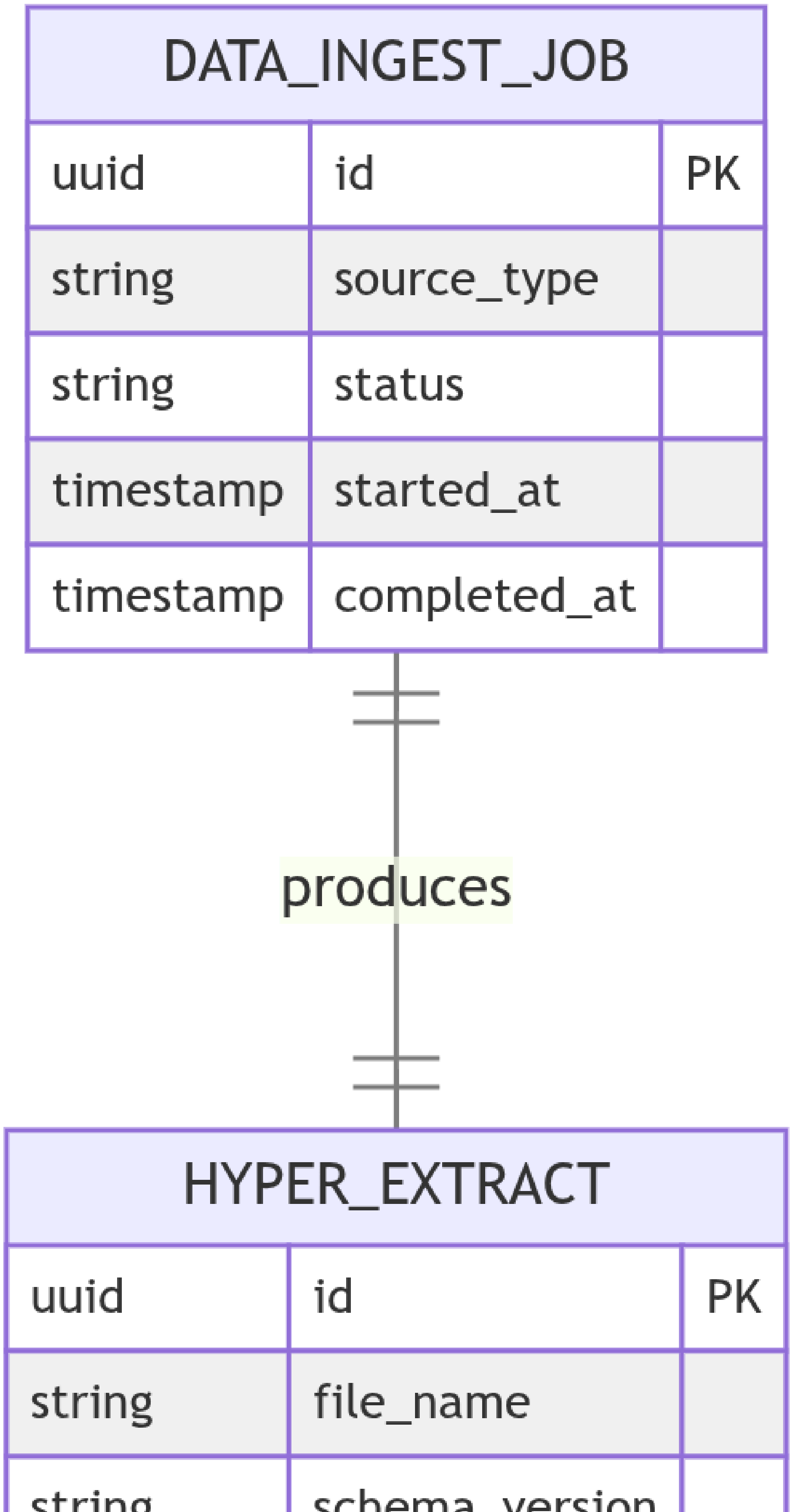
TABLEAU_WORKBOOK		
uuid	id	PK
string	tableau_id	

Why track Tableau metadata internally

- Embed dashboards dynamically
- Map RBAC ⟶ dashboard visibility
- Support the Metadata API lineage later

---

e. Hyper & Data Ingestion Tracking

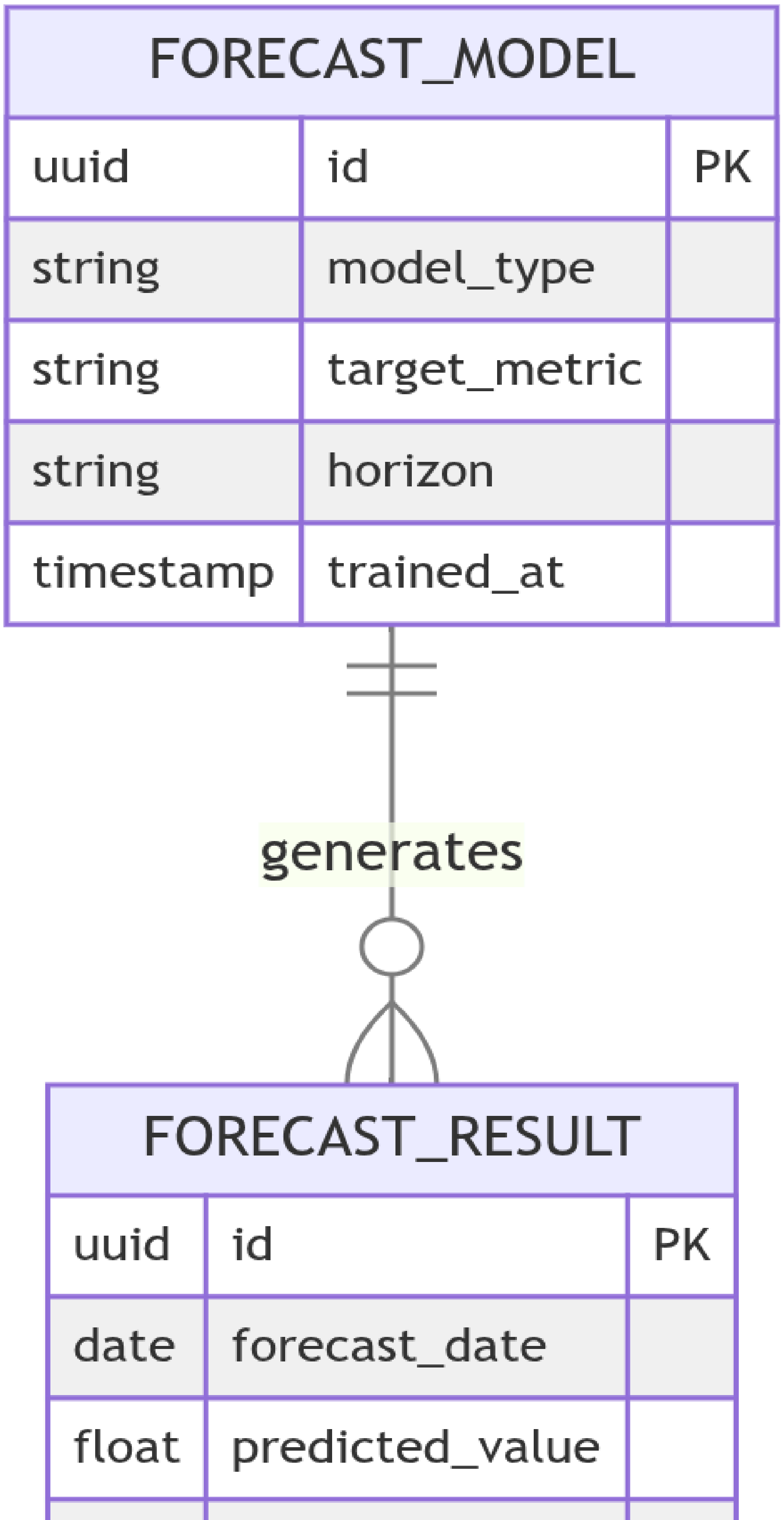

DATA_INGEST_JOB
uuid
id
PK
string
source_type
string
status
timestamp
started_at
timestamp
completed_at
produces
HYPER_EXTRACT
uuid
id
PK
string
file_name
string
schema_version

Used by:

- FastAPI ingestion service
- Retry & audit logs
- Hackathon demo observability

---

f. Predictive Analytics & Forecasting

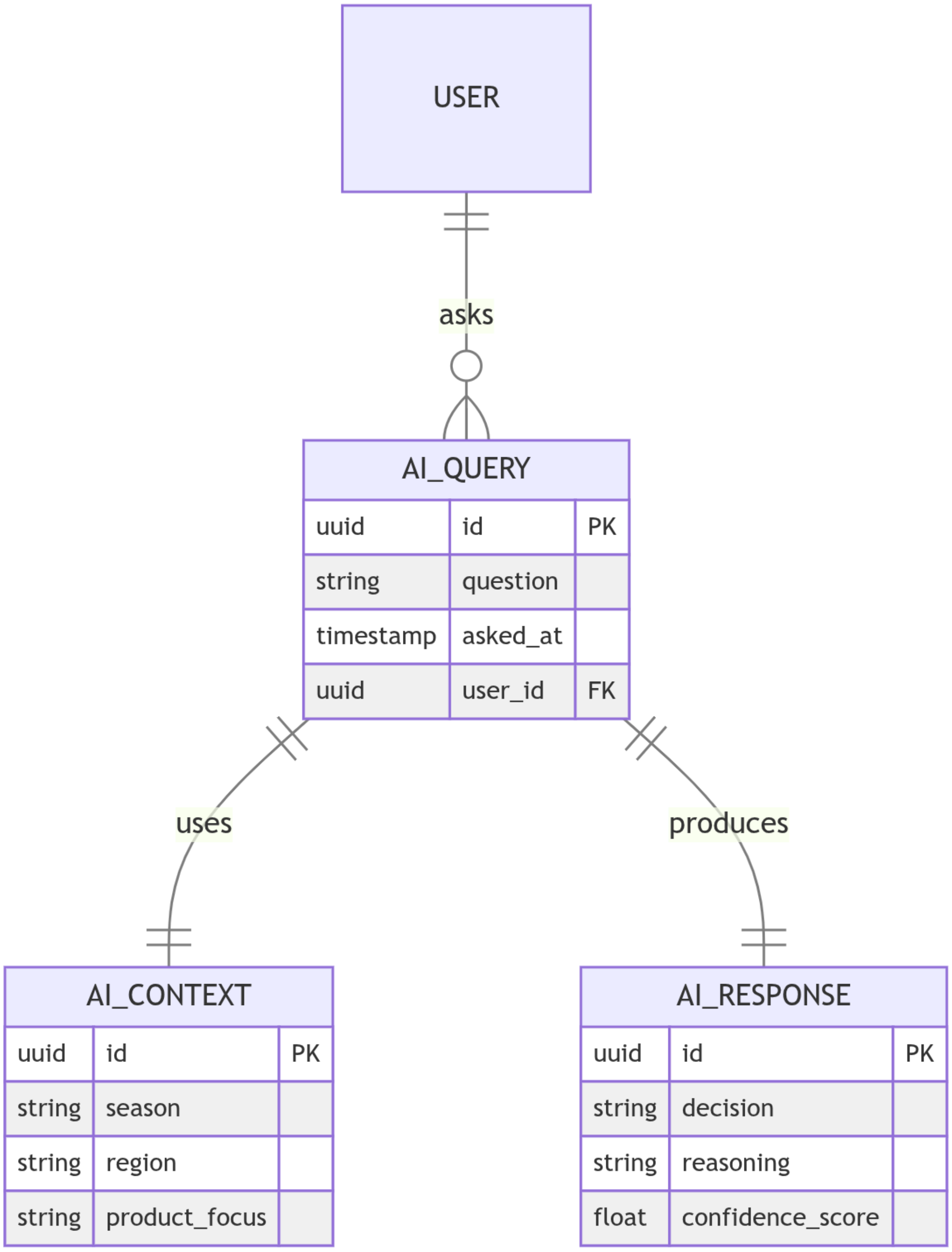

FORECAST_MODEL
uuid id PK
string model_type
string target_metric
string horizon
timestamp trained_at
generates
FORECAST_RESULT
uuid id PK
date forecast_date
float predicted_value

Can be:

- Tableau native forecasting
- TabPy Python models
- Future ML pipelines

---

g. LLM Decision Intelligence Layer

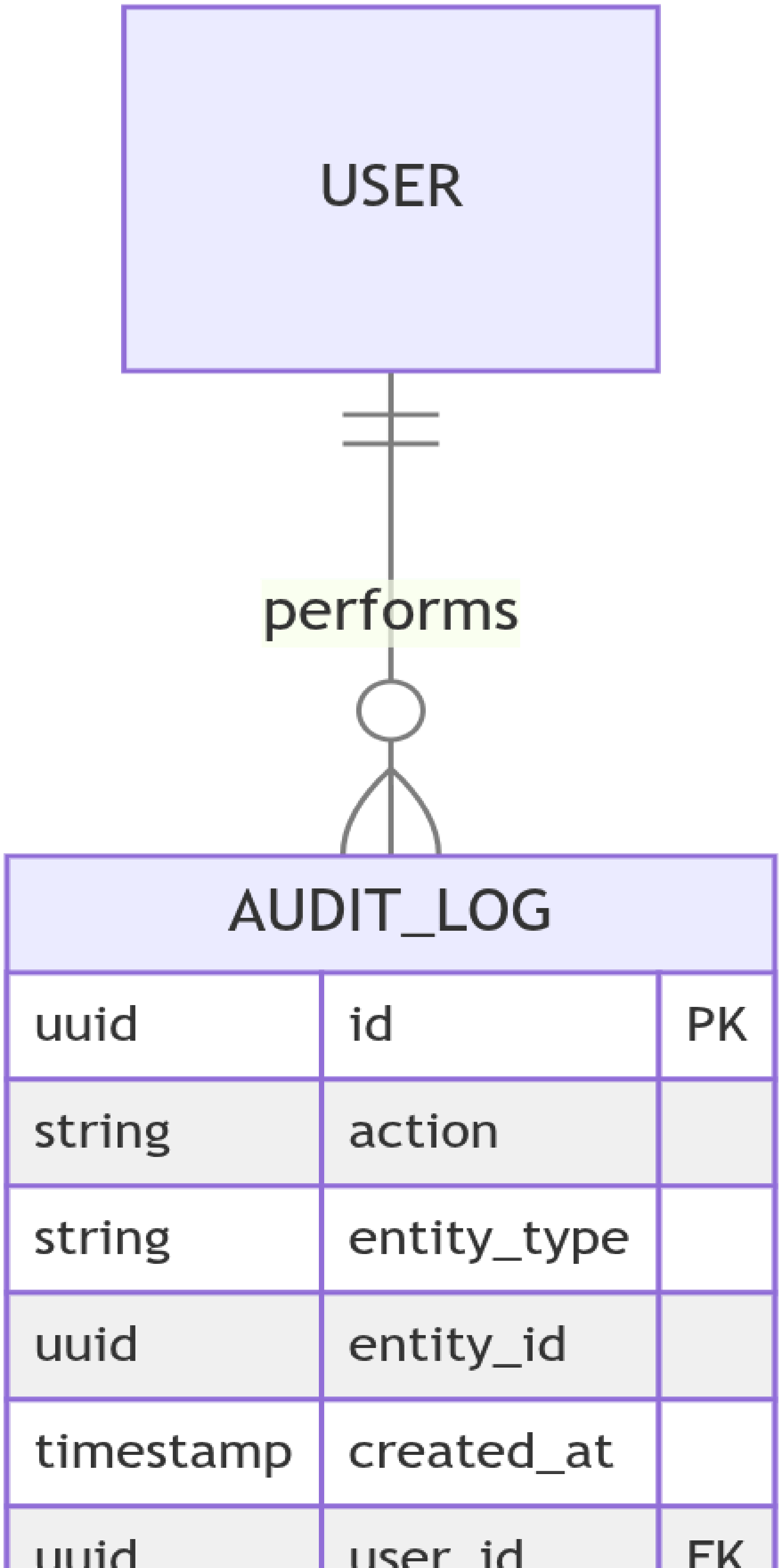


This is the killer differentiator

- Analytics ⟶ Context ⟶ Decision
- Not just dashboards, but **answers**

---

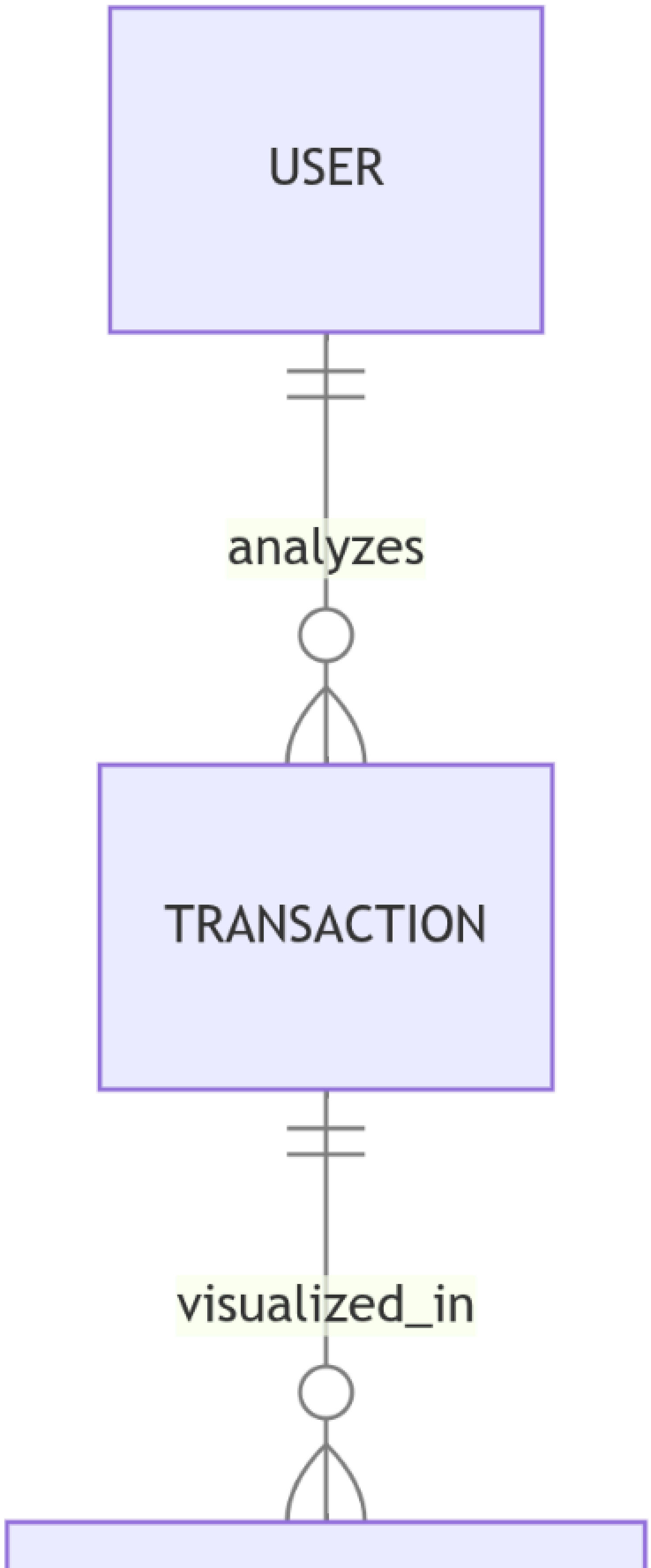
h. Audit & Observability


USER
performs
AUDIT_LOG
uuid
id
PK
string
action
string
entity_type
uuid
entity_id
timestamp
created_at
uuid
user_id
FK

Useful for:

- Hackathon demo trust
- Debugging ingestion & AI

How Everything Connects (Big Picture)


USER
analyzes
TRANSACTION
visualized_in

8. Data Ingestion Design

MVP Ingestion Flow

Why Hyper API?

- Optimized for analytics
- Incremental updates later
- Tableau-native format

9. Tableau API Usage Mapping

Requirement	Tableau API Used
Upload POS data	Hyper API + REST API
Publish dashboards	REST API
Embed dashboards	Tableau JS Embed
Predictive analytics	Built-in Forecast + TabPy
Metadata discovery	Metadata API (optional)
LLM integration	Extensions / Backend

10. Dashboard Design (MVP)

Dashboard 1: Sales Overview

- Total revenue
- Top products
- Sales by region

- Sales by demographic

Dashboard 2: Seasonal Trends

- Month vs product heatmap
- Category-wise seasonality

Dashboard 3: Predictive Insights

- Next 3-month sales forecast
- Product demand trend line

---

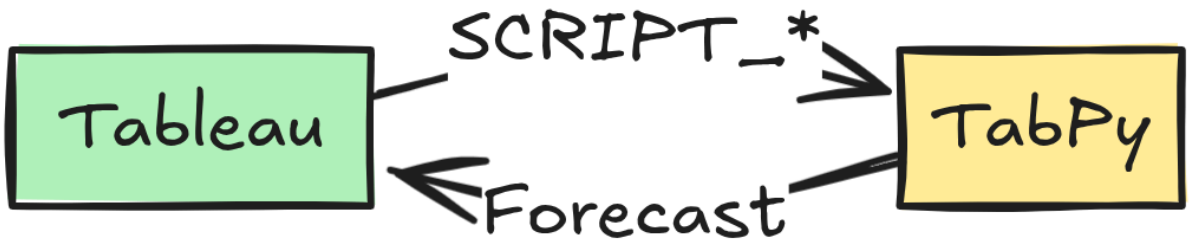
10. Predictive Analytics Design

Option A (MVP Preferred)

- Tableau built-in forecasting
- Fast, no infra overhead

Option B (Advanced MVP)

- TabPy-based Python model



---

11. LLM Decision Intelligence Layer

Purpose

Convert **analytics → decisions**

Example Questions

- *“Should I restock Product X?”*
- *“Which product will sell the most next winter?”*

LLM Input Structure

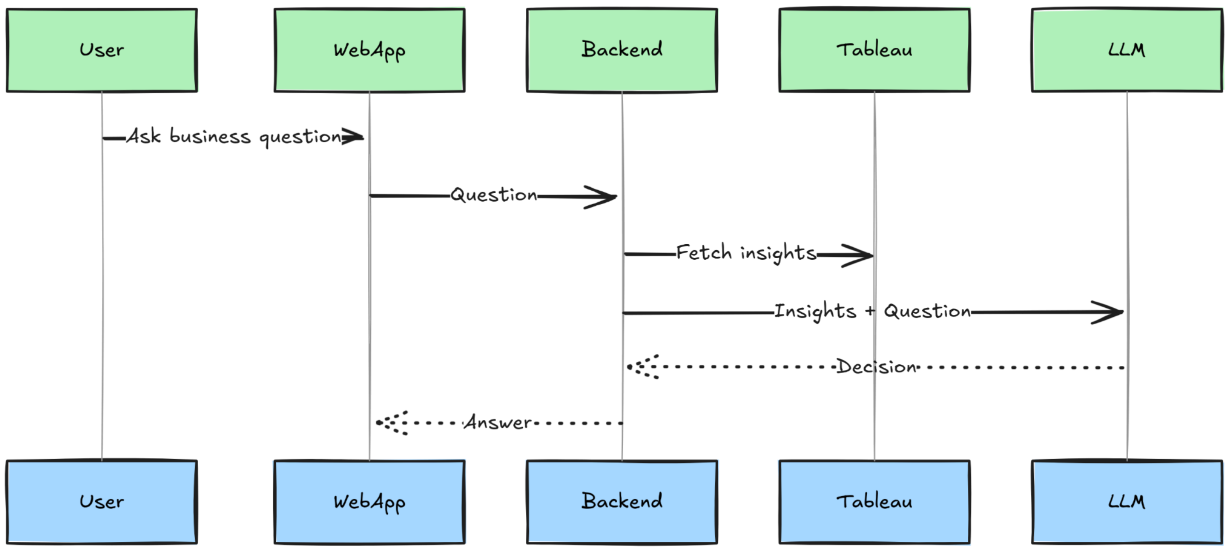
```
{
  "time_period": "Winter",
  "region": "Delhi",
  "top_products": ["Jackets", "Sweaters"],
  "sales_trend": "Increasing"
}
```

LLM Output

```
{
  "decision": "Stock winter jackets",
  "confidence": "High",
  "reason": "Consistent YoY growth in winter season"
}
```

---

11. LLM Integration Architecture (Basic Structure)



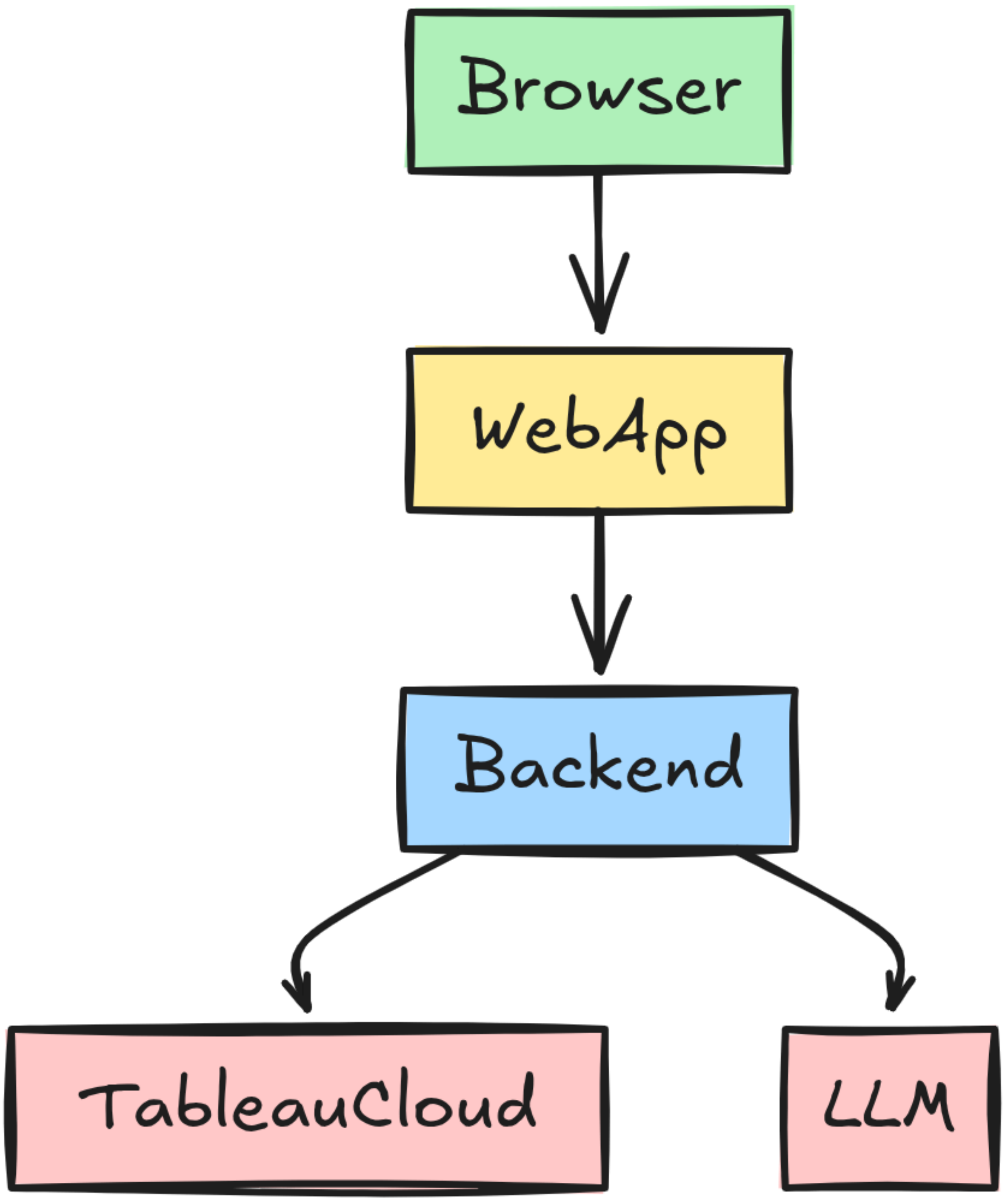
13. Embedded Analytics (SaaS UX)

- Dashboards embedded inside product UI
- Filters synced with SaaS controls
- Tableau remains “invisible” to the user

14. Security (MVP-Level)

- Read-only Tableau credentials
- Backend-only API keys
- No PII storage
- Sample data only

15. Deployment Model



---

16. Hackathon Judging Alignment

Criteria	How MVP Meets It
Innovation	Analytics → Decisions via LLM
Tableau Usage	Hyper, REST, TabPy, Embedding
Business Value	Retail inventory optimization
Extensibility	POS-agnostic architecture
Demo Readiness	Clear dashboards + AI output

---

17. MVP Demo Flow (Video Script)

1. Upload the POS dataset
2. Show Tableau dashboards
3. Apply filters (season, region)
4. Ask an AI question
5. Show an actionable decision

---

18. Future Roadmap (Post-Hackathon)

- Real-time POS ingestion (Kafka + incremental Hyper)
- Multi-store analytics
- Auto-retraining ML models
- Agentic workflows (auto reorder suggestions)
- Slack / Salesforce integration