Curated Usage Data API

API Interaction with Databases

The API primarily interacts with a PostgreSQL read replica.

- **Primary Database (user_activity):** This is a representation of the primary database responsible for all write operations, ensuring data consistency and integrity. It's the source of truth for all raw usage data.
- **Read Replica Database (user_activity_replica):** This instance is a representation of the a read replica that continuously replicates data from the primary and is configured as a read-only hot standby. The API's analytical queries (e.g., for daily active users) are directed to this replica to offload the primary, ensuring that heavy read workloads do not impact transactional performance.
- **Materialized View (daily_active_users):** Daily Active Users (DAU) is presented in a materialized view to precompute data for fast queries
- **User Privileges:** A dedicated replica_user is configured with SELECT privileges on the necessary tables (e.g., user_activity, daily_active_users) within the replica database, ensuring the API adheres to the principle of least privilege for read operations.

Authentication Strategy

The API employs an API key-based authentication strategy for securing its endpoints.

- **API Key Header:** Clients are required to provide an API key in the x-api-key HTTP header for authenticated endpoints.
- **Vault Integration:** The expected valid API key is securely loaded from HashiCorp Vault. This is a best practice for managing sensitive credentials, preventing hardcoding of secrets in the application code or environment variables.
- **Validation:** An authentication dependency (e.g., verify_api_key) intercepts incoming requests, validates the provided API key against the key retrieved from Vault, and returns a 401 Unauthorized error if the key is invalid or missing.

Caching

The Redis service is the caching mechanism.

- **Redis for Caching:** Redis serves as an in-memory cache layer for frequently accessed analytical queries (e.g., DAU totals for specific date ranges).
- **Cache-Aside Pattern:** The get_dau function checks Redis for cached results before querying the database. If a cache hit occurs, the cached data is returned directly. If a cache miss, the data is fetched from daily_active_users, and the result is then stored in Redis with an appropriate expiration time (SETEX).

Security & Privacy Considerations

- **Secrets Management:** HashiCorp Vault is used to store and retrieve sensitive API keys and database credentials.
- **Principle of Least Privilege:** The API connects to the replica database using a replica_user with only SELECT permissions, minimizing the potential impact of a security breach.
- **Network Segmentation:** Docker's internal networking (pgnet) isolates the database, Redis, Vault, and API services, limiting direct external exposure.
- **Data in Transit:** While the docker-compose.yml doesn't explicitly configure TLS for internal database/Redis connections, in a production environment, all communication (especially between the API and databases/Vault) would be encrypted using TLS/SSL.
- **Data at Rest:** PostgreSQL data is persisted using Docker volumes (primary_data, replica_data), ensuring data durability. Encryption at rest for these volumes would be a further enhancement in production.
- **Privacy (for Usage Data):** For usage data like DAU, it's crucial to ensure that the underlying raw data is either anonymized, aggregated, or de-identified to protect user privacy, especially if the API is exposed to external consumers.

Error Handling, Validation, and Logging

The API incorporates robust mechanisms for handling errors, validating inputs, and logging operational events.

- **Input Validation:**
  - FastAPI's Pydantic models automatically validate incoming request data (query parameters, request bodies) against defined schemas, ensuring data types and formats are correct.
  - Custom validation logic is implemented for business rules, such as date format (datetime.strptime) and logical constraints (e.g., end_date must be after start_date), raising HTTPException for invalid inputs.
- **Error Handling:**
  - FastAPI's HTTPException is used to return standardized HTTP error responses (e.g., 400 Bad Request, 401 Unauthorized, 404 Not Found, 500 Internal Server Error) with clear detail messages.
  - try-except blocks are used around critical operations (like database connections and queries) to catch exceptions, log them, and translate them into appropriate HTTPException responses.
- **Logging:**
  - A centralized logging setup (app.utils.logger.setup_logger()) is used to configure application-wide logging.
  - logging.getLogger(__name__) is used within modules to create context-aware loggers.
  - Error messages, database query failures, cache hits/misses, and other significant events are logged, providing observability and aiding in debugging and monitoring.