XenoFDE Documentation

Assumptions:

- The project presupposes a Shopify store that has API access and is using environment variables for SHOPIFY_STORE and SHOPIFY_ACCESS_TOKEN.
- Node.js and npm should be installed, and the dependencies must be listed in the package.json file.
- Prisma must be configured with a database (for example, SQLite or PostgreSQL), and migrations should be applied.
- The dashboard is a React application that uses Tailwind CSS and a modern browser is presumed.
- Webhooks are set up in Shopify to go to the server's endpoints.
- Data is being fetched every day through a cron job, thus it is assumed that the server is running continuously.
- Firebase is the chosen method for authentication in the dashboard and must be correctly configured.

Architecture Diagram:

```
[Shopify Store] --> [Express Server]
                 |
                 |--> Webhooks (cart abandoned, checkout started)
                 |
                 |--> API Endpoints (/api/metrics, /api/orders, etc.)
                 |
                 |--> Cron Job --> [fetchShopifyData.js] --> JSON Files (customers.json,
products.json, orders.json)
                 |
                 |--> [saveShopifyData.js] --> [Prisma Database]
                 |
[React Dashboard] <-- API Calls --> [Express Server]
```

Data Flow:

1. Shopify communicates webhooks to an Express server.
2. A cron job initiates fetchShopifyData.js to get data from the Shopify API and store it in JSON.
3. saveShopifyData.js upserts the data into Prisma DB.
4. The Dashboard uses API calls to the Express server to get metrics, orders, and customers.
5. The server accesses the JSON files or DB to send the response.

APIs:

- GET /: Health check, returns "Xeno Shopify Data Ingestion Service is running"
- POST /webhooks/cart/abandoned: Receives cart abandoned webhook, verifies HMAC, logs body.
- POST /webhooks/checkout/started: Receives checkout started webhook, verifies HMAC, logs body.
- GET /api/metrics: Returns totalCustomers, totalOrders, totalRevenue based on JSON data.
  Query params: start, end (dates)
- GET /api/orders: Returns filtered orders by date range.
  Query params: start, end
  Response: Array of {id, createdAt, totalPrice, customer, line_items}
- GET /api/customers/top: Returns top 5 customers by total spent.
- GET /api/customers: Returns raw customers data from JSON.
- GET /api/products: Returns raw products data from JSON.

Data Models (from Prisma schema):

- Tenant: id (String, primary), name (String)

- Customer: id (String, primary), tenantId (String, foreign), shopifyId (String, unique), email (String), firstName (String), lastName (String), createdAt (DateTime), updatedAt (DateTime)

- Product: id (String, primary), tenantId (String, foreign), shopifyId (String, unique), title (String), handle (String), productType (String), vendor (String), status (String), createdAt (DateTime), updatedAt (DateTime)

- ProductVariant: id (String, primary), productId (String, foreign), shopifyId (String, unique), title (String), sku (String), price (String), compareAtPrice (String?), inventoryQuantity (Int), createdAt (DateTime), updatedAt (DateTime)

- Order: id (String, primary), tenantId (String, foreign), shopifyId (String, unique), customerId (String, foreign), totalPrice (String), subtotalPrice (String), totalTax (String), createdAt (DateTime), updatedAt (DateTime)

- OrderLineItem: id (String, primary), orderId (String, foreign), productVariantId (String, foreign), quantity (Int), price (String), createdAt (DateTime), updatedAt (DateTime)

Next Steps

- Implement complete data saving: As it is now, only customers are saved to the DB; make products, orders, variants, line items be saved as well.
- Secure the API endpoints by adding authentication.
- Support webhooks and API calls with error handling, and also add logging.
- Add unit and integration tests for server endpoints and data fetching.
- Improve the dashboard speed, for example, use pagination for large datasets.
- Add more metrics and visualizations, for example, sales trends over time.
- Create a CI/CD pipeline for automated deployment.
- Add environment-specific configurations (development, production).
- Incoming data validation and sanitization should be implemented.
- Add user roles and permissions in the dashboard.
- Transition to full DB usage instead of JSON files for better performance and reliability.