Shopify Data Ingestion – Documentation

Known Assumptions & Limitations

Multi-Tenancy

- Tenant isolation is implemented using a tenant_id column on each table.
- This strategy is straightforward but less scalable compared to schema-per-tenant or database-per-tenant models.
- Designed for a relatively small number of tenants during the prototype stage.

Shopify Integration

- Only Customers, Orders, Products, Carts, and Checkouts are ingested.
- Bulk ingestion through Shopify APIs has not been implemented; the system relies on real-time webhooks and basic ingestion endpoints.
- Assumes webhook delivery is reliable — no retry or backoff logic is in place.

Authentication

- Uses basic email/password authentication for tenant login.
- JWT-based authentication and role-based access control are not yet implemented.
- Assumes a small set of internal or test users.

Data Quality & Synchronization

- Shopify is treated as the source of truth for data.
- If webhook events are missed, manual re-sync APIs would be required (currently not implemented).
- Assumes consistency between Shopify and the local database.

Cron Jobs & Scheduling

- Uses node-cron for simulating cart and checkout abandonment detection.
- The cutoff window is fixed (e.g., 5 minutes) rather than configurable per tenant.
- No distributed scheduler is set up — assumes a single-instance deployment.

Dashboard & Insights

- Displays basic metrics such as total customers, orders, and revenue, with simple charts.
- Only the top 5 customers by spend are shown.
- Date filtering is implemented only for orders.
- Advanced analytics such as CLV and RFM segmentation are not yet included.

Deployment

- Deployed to Render/Vercel free tiers, suitable for demo purposes only.
- CI/CD pipelines have not been set up.
- Environment variables are stored in a .env file rather than a secrets manager.

Monitoring & Logging

- Limited to basic console logging.
- No centralized monitoring or alerting tools (e.g., Datadog, Prometheus) are integrated.
- Assumes low traffic and minimal operational load.

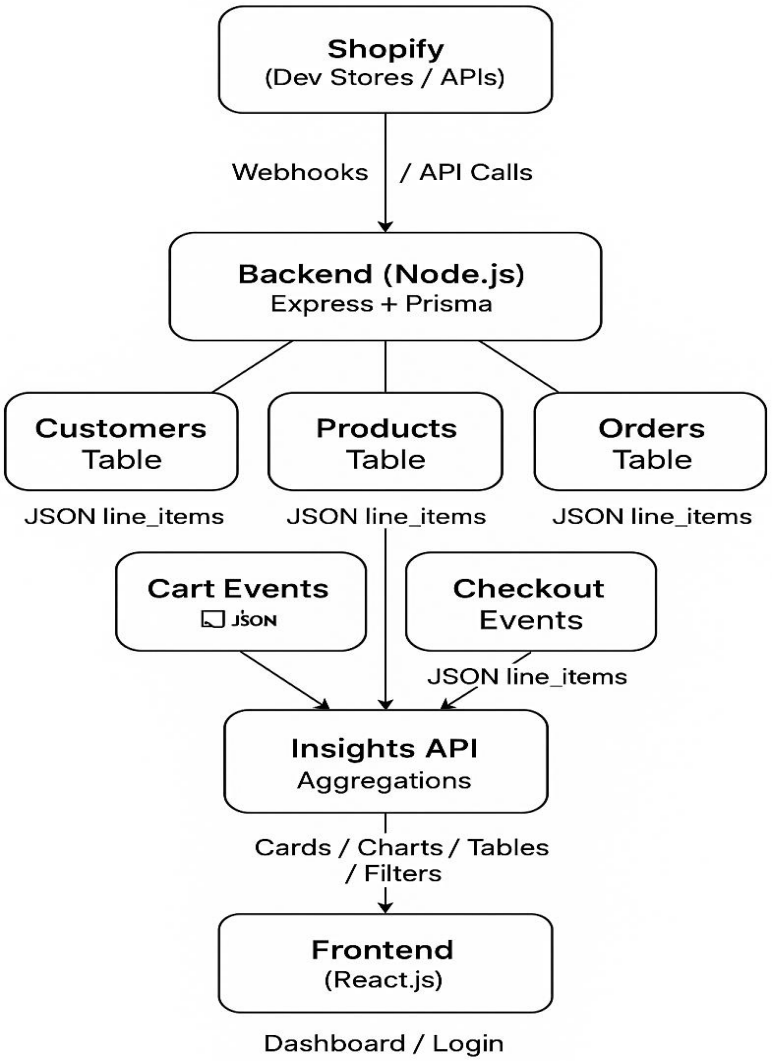
Performance

- Database indexing is minimal, with only essential unique constraints in place.
- No caching layer (e.g., Redis) has been added.
- Designed for a modest dataset during the demo phase.

2. High-Level Architecture

The system ingests data from Shopify via webhooks, stores it in a multi-tenant database, and provides a dashboard for analytics.

Architecture Diagram:



3. APIs and Data Models

Webhook Endpoints:

- POST /webhooks/customers
- POST /webhooks/products
- POST /webhooks/orders
- POST /webhooks/carts
- POST /webhooks/checkouts

Ingestion Endpoints:

- POST /ingest/customers
- POST /ingest/products
- POST /ingest/orders

Tenant & Dashboard APIs:

- POST /tenants/register, /login
- GET /api/customers, /products, /orders, /insights

Key Data Models (Prisma):

- **tenants:** id, store_url, webhook_secret, email, name
- **customers:** id, tenant_id, shopify_id, name, email
- **products:** id, tenant_id, shopify_id, title, price
- **orders:** id, tenant_id, shopify_id, status, total_price
- **carts:** id, tenant_id, shopify_id, status
- **checkouts:** id, tenant_id, shopify_id, status

4. Next Steps to Productionize

Robust Multi-Tenant Support

- Fully isolate tenant data at the database level to prevent cross-tenant access.
- Enforce tenant-specific API keys and secrets for added security.
- Provide onboarding APIs that allow tenants to self-register and configure their stores.

Webhook Reliability

- Implement retry logic with exponential backoff for failed webhook deliveries.
- Store raw webhook payloads in a message queue (e.g., RabbitMQ, Kafka) for asynchronous processing.
- Add HMAC signature verification to ensure webhook authenticity in production.

Scalability & Performance

- Introduce Redis caching for frequently accessed metrics (e.g., total revenue, top customers).
- Use background workers for resource-intensive tasks (e.g., generating insights or reports).
- Consider partitioning or sharding data by tenant if database size grows significantly.

Data Quality & Synchronization

- Add full synchronization jobs to complement webhooks and recover missed events.
- Implement deduplication logic to prevent duplicate records during ingestion.

Security

- Store all secrets (API keys, webhook secrets) in a secure vault (e.g., AWS Secrets Manager, HashiCorp Vault).
- Enforce HTTPS across all environments.
- Introduce role-based access control (RBAC) for tenants and dashboard users.

Dashboard Enhancements

- Expand analytics with visualizations such as cohort analysis and customer lifetime value (CLV).
- Enable real-time updates via WebSockets or Server-Sent Events.
- Add filters for date ranges, customer segments, and product categories.

Monitoring & Observability

- Use structured logging for webhook and API calls.
- Integrate with monitoring platforms like Grafana or Prometheus for metrics and dashboards.
- Set up alerts for ingestion failures, high latency, or other anomalies.

Deployment & CI/CD

- Containerize the application using Docker and deploy to scalable platforms (e.g., Kubernetes, AWS ECS).
- Automate database migrations and seed scripts with Prisma Migrate.
- Build comprehensive test suites (unit, integration, and load tests) to ensure system stability.

Compliance

- Guarantee tenant data isolation and adhere to privacy regulations (e.g., GDPR, CCPA).
- Define and enforce data retention policies for customer PII.