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.

## 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.

## 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.

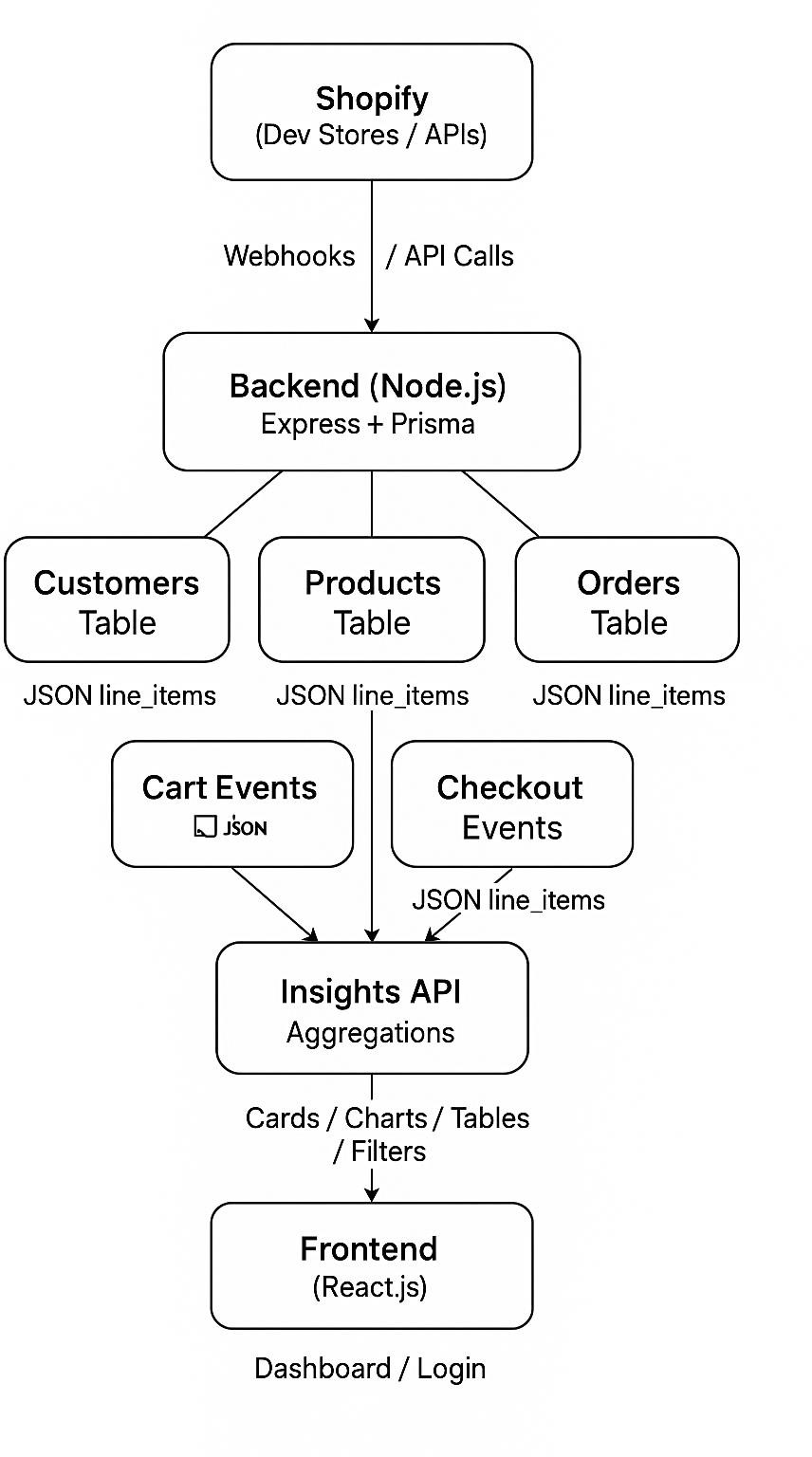
## Performance

* Database indexing is minimal, with only essential unique constraints in place.
* Designed for a modest dataset during the demo phase.

# 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:**



# APIs and Data Models

### Webhook Endpoints:

* + POST /webhooks/customers
  + POST /webhooks/products
  + POST /webhooks/orders
  + POST /webhooks/carts
  + POST /webhooks/checkouts

### 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

# 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.

## Scalability & Performance

* + 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.

## 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.

## 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).
  + Build comprehensive test suites (unit, integration, and load tests) to ensure system stability.