Xeno FDE Internship: Project Documentation

Project: Multi-Tenant Shopify Data Ingestion & Insights Service

Author: Tanisha Ranjan(22BCE7027), Vellore Institute of Technology - Amaravati

1. Assumptions & Trade-offs

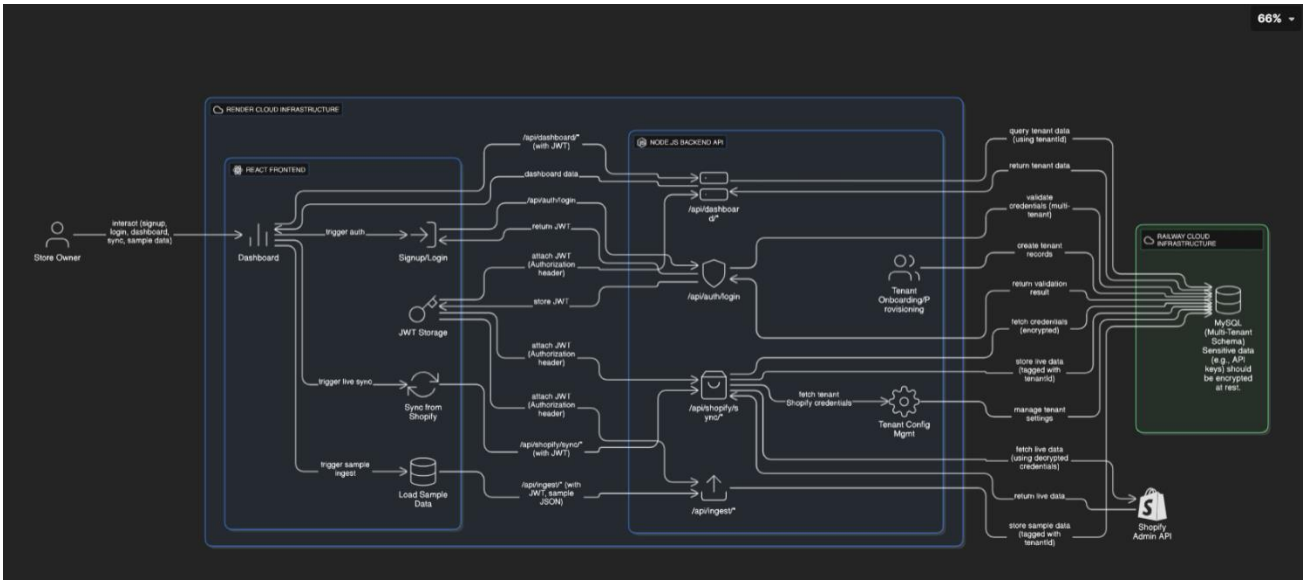
This project was developed as a proof-of-concept to demonstrate the core functionalities of a multi-tenant data platform. The following assumptions and trade-offs were made to meet the assignment's scope and deadline:

- Authentication & Authorization: A simple email and password system is used for tenant onboarding. For API security, a JWT (JSON Web Token) is issued upon login and used to authorize all subsequent requests. The user's tenantId is securely extracted from this token on the backend to ensure strict data isolation.
- Shopify Data Sync: The application connects to the live Shopify Admin API to sync products, customers, and orders. For simplicity and to avoid a full OAuth flow, the Shopify Access Token is collected during signup and stored. In a production scenario, this would be replaced with a secure OAuth 2.0 handshake.
- Data Ingestion: The "Load Sample Data" feature uses the application's own ingestion API to populate the database with a static, pre-defined dataset. This provides a reliable baseline for demonstrating the dashboard's features without requiring a live Shopify connection.
- Background Tasks: The project includes a simple scheduler stub (scheduler.js) to demonstrate the capability for background data synchronization. For the final working demo, this was disabled in favor of manual sync buttons to prevent data conflicts and provide a clearer user experience.

2. High-Level Architecture

The application follows a modern three-tier architecture, with separate services forthe database, backend API, and frontend user interface, all deployed on cloud platforms.

Architecture Diagram



- **Frontend:** A single-page application built with React, deployed as a Static Site on Render. It handles user authentication and renders all the data visualizations.
- **Backend:** A Node.js and Express API server deployed as a Web Service on Render. It manages all business logic, including user authentication, multi-tenant data isolation, and communication with the Shopify API.
- **Database:** A MySQL database hosted on Railway. It stores all tenant information and their associated e-commerce data. Sequelize is used as the ORM to manage the database schema and queries.

3.APIs and Data Models

Method	Endpoint	Description
POST	auth/signup	Creates a new tenant account.
POST	auth/login	Authenticates a tenant and returns a JWT.
POST	ingest/products	Ingests a list of sample products for the logged-in tenant.
POST	ingest/customers	Ingests a list of sample customers.
POST	ingest/orders	Ingests a list of sample orders and links them to products/customers.
DELETE	ingest/tenant/all-data	Clears all sample data for the logged-in tenant.
GET	dashboard/tenant-info	Fetches the name, email, and logo for the logged-in tenant.
GET	dashboard/overview	Fetches high-level metrics (total revenue, orders, customers).I
GET	dashboard/sales-trend	Fetches data for the 30-day sales trend chart.
GET	dashboard/top-customers-chart	Fetches data for the top 5 customers chart.
GET	dashboard/orders	Fetches a paginated list of orders with product and customer details.
GET	/customer-history	Fetches a list of all customers with their associated orders.

Database Schema

The database uses a multi-tenant model where data is isolated by a tenantId.

- Tenant: Stores information about each registered store, including their name, URL, and credentials.
- Customer: Stores customer information. A composite unique key on (tenantId, shopifyId) ensures customer IDs are unique per store.
- Product: Stores product information, with a composite unique key on (tenantId, shopifyId).
- Order: Stores order information, with a composite unique key on (tenantId, shopifyId).
- OrderItem: A join table that links products to orders in a many-to-many relationship.

4. Next Steps to Productionize

This application provides a strong foundation. To make it a production-ready, enterprise-grade service, the following steps would be necessary:

Enhanced Security:

- Encrypt Access Tokens: The Shopify access tokens must be encrypted at rest in the database using a strong algorithm (e.g., AES-256) and a secret key stored securely in the environment.
- Add Production-Grade Auth: Implement features like email verification, password reset, and two-factor authentication (2FA) for tenant accounts.

Scalability & Robustness:

- Background Job Queues: Move all data ingestion and synchronization tasks to a background job queue (e.g., using Redis and BullMQ). This prevents API requests from timing out on large syncs and allows for automatic retries on failed jobs.
- CI/CD Pipeline: Create a Continuous Integration/Continuous Deployment pipeline (e.g., with GitHub Actions) to automate testing and deployments, ensuring code quality and faster release cycles.

Monitoring & Testing:

- Application Monitoring: Integrate a monitoring service (like Datadog or Sentry) to track API performance, error rates, and set up alerts for critical issues.
- Automated Testing: Write a full suite of unit, integration, and end-to-end tests to ensure the reliability and correctness of all features.

5. You can access the live, deployed application at the following links:

- **Frontend (Live Site):** https://shopify-store-frontend.onrender.com
- **Backend API:** https://shopify-store-backend-r8t7.onrender.com

Note: The free-tier services may take 30-60 seconds to "wake up" on the first visit.