**GlobalPulse MVP Specification Report (GMSR)**

**1. Project Identity**

* **Name**: GlobalPulse MVP
* **Purpose**: Develop a Minimum Viable Product (MVP) to ingest global economic data from Trading Economics, transform it into a canonical format, forecast indicators, map them to a predefined taxonomy, and visualize results in a responsive React dashboard.
* **High-level Scope**:
  + Ingest raw economic data (countries, indicators, timeseries) via Trading Economics API.
  + Transform and validate data into canonical tables (canonical\_countries, canonical\_indicators, canonical\_timeseries).
  + Forecast economic indicators using Python (statsmodels).
  + Map indicators to 5 economic categories (Growth, Prices, Labor, Trade, Sentiment).
  + Build a modular React dashboard with TypeScript and ESM for visualization.
  + Use Supabase for data storage, TypeScript/Node.js for backend/frontend, Python for forecasting.
  + Ensure modular, additive design to prevent regressions.
* **Target Audience / Constraints**:
  + Audience: Economists, analysts, policymakers requiring real-time economic insights.
  + Constraints: Windows 11, Lenovo IdeaPad 330S-141KB, Supabase database, TypeScript/Node.js for backend/frontend, Python for forecasting, modular architecture, scalable testing.

**2. Chronological Milestone Ledger**

**Milestone 1: Phase 1, Stage 1 - Project Setup and Environment Configuration**

* **✅ Status**: Completed
* **Deliverables Snapshot**:
  + **File Structure**:
  + E:\Users\Public\globalpulse
  + ├── .env
  + ├── dashboard
  + │ ├── package.json
  + │ ├── node\_modules
  + ├── forecasting
  + │ ├── .env
  + │ ├── requirements.txt
  + │ ├── venv
  + ├── db
  + │ ├── migrations
  + │ ├── queries
  + **Files**:
    - globalpulse/.env:
    - SUPABASE\_URL=https://jjizaincjbgigbylbbyi.supabase.co
    - SUPABASE\_SERVICE\_ROLE\_KEY=your-service-role-key-here
    - TRADING\_ECONOMICS\_API\_KEY=your-te-api-key-here
    - globalpulse/forecasting/.env:
    - SUPABASE\_URL=https://jjizaincjbgigbylbbyi.supabase.co
    - SUPABASE\_SERVICE\_ROLE\_KEY=your-service-role-key-here
    - TRADING\_ECONOMICS\_API\_KEY=your-te-api-key-here
    - globalpulse/forecasting/requirements.txt:
    - pandas==2.2.2
    - numpy==1.26.4
    - supabase==2.7.4
    - python-dotenv==1.0.1
    - statsmodels==0.14.2
    - requests==2.32.3
    - globalpulse/dashboard/package.json:
    - {
    - "name": "globalpulse-dashboard",
    - "version": "1.0.0",
    - "type": "module",
    - "scripts": {
    - "start": "node scripts/ingest\_te\_indicators.js"
    - },
    - "dependencies": {
    - "@supabase/supabase-js": "^2.45.4",
    - "dotenv": "^16.4.5"
    - }
    - }
  + **Verification & Test Criteria**:
    - Python virtual environment activated: .\venv\Scripts\activate.
    - Dependencies installed: pip install -r forecasting/requirements.txt.
    - Node.js dependencies installed: cd dashboard && npm install.
    - Environment variables set in .env files.
    - Test: pip list shows required Python packages; npm list shows @supabase/supabase-js, dotenv.

**Milestone 2: Phase 2, Stage 1 - Database Schema and Migrations**

* **✅ Status**: Completed
* **Deliverables Snapshot**:
  + **File Structure**:
  + E:\Users\Public\globalpulse\db
  + ├── migrations
  + │ ├── 001\_create\_tables.sql
  + │ ├── 002\_create\_rls\_policies.sql
  + │ ├── 003\_clear\_tables.sql
  + │ ├── 004\_create\_validation\_functions.sql
  + │ ├── 005\_create\_validation\_log.sql
  + │ ├── 006\_create\_forecast\_results.sql
  + │ ├── 007\_create\_taxonomy\_tables.sql
  + │ ├── 008\_seed\_categories.sql
  + ├── queries
  + │ ├── check\_duplicates.sql
  + │ ├── diagnose\_validation\_failure.sql
  + │ ├── diagnose\_timeseries\_count.sql
  + │ ├── test\_validation\_edge\_cases.sql
  + │ ├── verify\_data.sql
  + │ ├── verify\_validation.sql
  + │ ├── verify\_preprocess\_log.sql
  + │ ├── verify\_forecast\_results.sql
  + │ ├── verify\_forecast\_log.sql
  + **Files**:
    - db/migrations/001\_create\_tables.sql:
    - CREATE TABLE raw\_countries (
    - country\_id TEXT PRIMARY KEY,
    - country\_name TEXT NOT NULL,
    - ingestion\_date TIMESTAMP WITH TIME ZONE DEFAULT NOW()
    - );
    - CREATE TABLE raw\_indicators (
    - indicator\_id TEXT PRIMARY KEY,
    - indicator\_name TEXT NOT NULL,
    - ingestion\_date TIMESTAMP WITH TIME ZONE DEFAULT NOW()
    - );
    - CREATE TABLE raw\_timeseries (
    - timeseries\_id SERIAL PRIMARY KEY,
    - country\_id TEXT REFERENCES raw\_countries(country\_id),
    - indicator\_id TEXT REFERENCES raw\_indicators(indicator\_id),
    - date DATE NOT NULL,
    - value NUMERIC NOT NULL,
    - ingestion\_date TIMESTAMP WITH TIME ZONE DEFAULT NOW()
    - );
    - CREATE TABLE canonical\_countries (
    - country\_code TEXT PRIMARY KEY,
    - country\_name TEXT NOT NULL,
    - ingestion\_date TIMESTAMP WITH TIME ZONE DEFAULT NOW()
    - );
    - CREATE TABLE canonical\_indicators (
    - indicator\_id SERIAL PRIMARY KEY,
    - indicator\_name TEXT NOT NULL,
    - ingestion\_date TIMESTAMP WITH TIME ZONE DEFAULT NOW()
    - );
    - CREATE TABLE canonical\_timeseries (
    - timeseries\_id SERIAL PRIMARY KEY,
    - country\_code TEXT REFERENCES canonical\_countries(country\_code),
    - indicator\_id INTEGER REFERENCES canonical\_indicators(indicator\_id),
    - date DATE NOT NULL,
    - value NUMERIC NOT NULL,
    - ingestion\_date TIMESTAMP WITH TIME ZONE DEFAULT NOW()
    - );
    - db/migrations/002\_create\_rls\_policies.sql:
    - ALTER TABLE raw\_countries ENABLE ROW LEVEL SECURITY;
    - ALTER TABLE raw\_indicators ENABLE ROW LEVEL SECURITY;
    - ALTER TABLE raw\_timeseries ENABLE ROW LEVEL SECURITY;
    - ALTER TABLE canonical\_countries ENABLE ROW LEVEL SECURITY;
    - ALTER TABLE canonical\_indicators ENABLE ROW LEVEL SECURITY;
    - ALTER TABLE canonical\_timeseries ENABLE ROW LEVEL SECURITY;
    - CREATE POLICY anon\_select\_raw\_countries ON raw\_countries FOR SELECT USING (true);
    - CREATE POLICY anon\_select\_raw\_indicators ON raw\_indicators FOR SELECT USING (true);
    - CREATE POLICY anon\_select\_raw\_timeseries ON raw\_timeseries FOR SELECT USING (true);
    - CREATE POLICY anon\_select\_canonical\_countries ON canonical\_countries FOR SELECT USING (true);
    - CREATE POLICY anon\_select\_canonical\_indicators ON canonical\_indicators FOR SELECT USING (true);
    - CREATE POLICY anon\_select\_canonical\_timeseries ON canonical\_timeseries FOR SELECT USING (true);
    - db/migrations/003\_clear\_tables.sql:
    - TRUNCATE TABLE raw\_timeseries, raw\_indicators, raw\_countries,
    - canonical\_timeseries, canonical\_indicators, canonical\_countries
    - RESTART IDENTITY CASCADE;
    - db/migrations/004\_create\_validation\_functions.sql:
    - CREATE OR REPLACE FUNCTION validate\_timeseries()
    - RETURNS TRIGGER AS $$
    - BEGIN
    - IF NEW.value IS NULL THEN
    - RAISE EXCEPTION 'Value cannot be null';
    - END IF;
    - IF NEW.date IS NULL THEN
    - RAISE EXCEPTION 'Date cannot be null';
    - END IF;
    - RETURN NEW;
    - END;
    - $$ LANGUAGE plpgsql;
    - CREATE TRIGGER timeseries\_validation
    - BEFORE INSERT OR UPDATE ON canonical\_timeseries
    - FOR EACH ROW EXECUTE FUNCTION validate\_timeseries();
    - db/migrations/005\_create\_validation\_log.sql:
    - CREATE TABLE validation\_log (
    - log\_id SERIAL PRIMARY KEY,
    - table\_name TEXT NOT NULL,
    - record\_id INTEGER,
    - error\_message TEXT NOT NULL,
    - validation\_date TIMESTAMP WITH TIME ZONE DEFAULT NOW()
    - );
    - db/migrations/006\_create\_forecast\_results.sql:
    - CREATE TABLE forecast\_results (
    - forecast\_id SERIAL PRIMARY KEY,
    - country\_code TEXT REFERENCES canonical\_countries(country\_code),
    - indicator\_id INTEGER REFERENCES canonical\_indicators(indicator\_id),
    - forecast\_date TIMESTAMP WITH TIME ZONE NOT NULL,
    - forecast\_values NUMERIC[] NOT NULL
    - );
    - ALTER TABLE forecast\_results ENABLE ROW LEVEL SECURITY;
    - CREATE POLICY anon\_select\_forecast\_results ON forecast\_results FOR SELECT USING (true);
    - db/migrations/007\_create\_taxonomy\_tables.sql:
    - CREATE TABLE categories (
    - category\_id SERIAL PRIMARY KEY,
    - category\_name TEXT NOT NULL UNIQUE,
    - created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
    - );
    - CREATE TABLE taxonomy\_mapping (
    - mapping\_id SERIAL PRIMARY KEY,
    - country\_code TEXT REFERENCES canonical\_countries(country\_code),
    - indicator\_id INTEGER REFERENCES canonical\_indicators(indicator\_id),
    - category\_id INTEGER REFERENCES categories(category\_id),
    - rank INTEGER NOT NULL,
    - is\_primary BOOLEAN NOT NULL,
    - fallback\_reason TEXT,
    - coverage\_ratio NUMERIC NOT NULL,
    - provenance JSONB NOT NULL,
    - mapping\_date TIMESTAMP WITH TIME ZONE DEFAULT NOW()
    - );
    - ALTER TABLE categories ENABLE ROW LEVEL SECURITY;
    - ALTER TABLE taxonomy\_mapping ENABLE ROW LEVEL SECURITY;
    - CREATE POLICY anon\_select\_categories ON categories FOR SELECT USING (true);
    - CREATE POLICY anon\_select\_taxonomy\_mapping ON taxonomy\_mapping FOR SELECT USING (true);
    - db/migrations/008\_seed\_categories.sql:
    - INSERT INTO categories (category\_name) VALUES
    - ('Growth'),
    - ('Prices'),
    - ('Labor'),
    - ('Trade'),
    - ('Sentiment');
    - db/queries/check\_duplicates.sql:
    - SELECT country\_code, indicator\_id, date, COUNT(\*)
    - FROM canonical\_timeseries
    - GROUP BY country\_code, indicator\_id, date
    - HAVING COUNT(\*) > 1;
    - db/queries/diagnose\_validation\_failure.sql:
    - SELECT table\_name, error\_message, validation\_date
    - FROM validation\_log
    - WHERE validation\_date > NOW() - INTERVAL '1 day';
    - db/queries/diagnose\_timeseries\_count.sql:
    - SELECT country\_code, indicator\_id, COUNT(\*) as record\_count
    - FROM canonical\_timeseries
    - GROUP BY country\_code, indicator\_id
    - ORDER BY record\_count DESC;
    - db/queries/test\_validation\_edge\_cases.sql:
    - INSERT INTO canonical\_timeseries (country\_code, indicator\_id, date, value)
    - VALUES ('INVALID', 999, '2023-01-01', 100)
    - ON CONFLICT DO NOTHING
    - RETURNING \*;
    - db/queries/verify\_data.sql:
    - SELECT COUNT(\*) FROM canonical\_countries WHERE country\_name IS NOT NULL;
    - SELECT COUNT(\*) FROM canonical\_indicators WHERE indicator\_name IS NOT NULL;
    - SELECT COUNT(\*) FROM canonical\_timeseries WHERE value IS NOT NULL AND date IS NOT NULL;
    - db/queries/verify\_validation.sql:
    - SELECT COUNT(\*) FROM validation\_log;
    - db/queries/verify\_preprocess\_log.sql:
    - SELECT \* FROM validation\_log WHERE table\_name = 'canonical\_timeseries';
    - db/queries/verify\_forecast\_results.sql:
    - SELECT country\_code, indicator\_id, forecast\_date
    - FROM forecast\_results
    - WHERE forecast\_date > NOW() - INTERVAL '1 day';
    - db/queries/verify\_forecast\_log.sql:
    - SELECT \* FROM validation\_log WHERE table\_name = 'forecast\_results';
  + **Verification & Test Criteria**:
    - Migrations applied: psql -h jjizaincjbgigbylbbyi.supabase.co -U postgres -d postgres -f db/migrations/001\_create\_tables.sql, etc.
    - Tables created: raw\_countries, raw\_indicators, raw\_timeseries, canonical\_countries, canonical\_indicators, canonical\_timeseries, categories, taxonomy\_mapping, forecast\_results.
    - Categories seeded: psql -c "SELECT COUNT(\*) FROM categories;" returns 5.
    - RLS policies enabled: SELECT \* FROM pg\_policies; shows policies for all tables.

**Milestone 3: Phase 2, Stage 2 - Data Ingestion and Transformation**

* **✅ Status**: Completed
* **Deliverables Snapshot**:
  + **File Structure**:
  + E:\Users\Public\globalpulse\dashboard\scripts
  + ├── ingest\_te\_indicators.js
  + ├── ingest\_te\_timeseries.js
  + ├── test\_supabase\_connection.js
  + ├── transform\_to\_canonical.js
  + ├── validate\_data.js
  + **Files**:
    - dashboard/scripts/ingest\_te\_indicators.js:
    - import { createClient } from '@supabase/supabase-js';
    - import dotenv from 'dotenv';
    - dotenv.config();
    - const supabase = createClient(process.env.SUPABASE\_URL, process.env.SUPABASE\_SERVICE\_ROLE\_KEY);
    - async function ingestTEIndicators() {
    - const response = await fetch('https://api.tradingeconomics.com/indicators', {
    - headers: { Authorization: `Bearer ${process.env.TRADING\_ECONOMICS\_API\_KEY}` }
    - });
    - const indicators = await response.json();
    - await supabase.from('raw\_indicators').insert(indicators.map(i => ({
    - indicator\_id: i.id,
    - indicator\_name: i.name,
    - ingestion\_date: new Date().toISOString()
    - })));
    - console.log(`Ingested ${indicators.length} indicators`);
    - }
    - ingestTEIndicators().catch(console.error);
    - dashboard/scripts/ingest\_te\_timeseries.js:
    - import { createClient } from '@supabase/supabase-js';
    - import dotenv from 'dotenv';
    - dotenv.config();
    - const supabase = createClient(process.env.SUPABASE\_URL, process.env.SUPABASE\_SERVICE\_ROLE\_KEY);
    - async function ingestTETimeseries() {
    - const { data: indicators } = await supabase.from('raw\_indicators').select('indicator\_id');
    - const { data: countries } = await supabase.from('raw\_countries').select('country\_id');
    - for (const country of countries) {
    - for (const indicator of indicators) {
    - const response = await fetch(
    - `https://api.tradingeconomics.com/historical/country/${country.country\_id}/indicator/${indicator.indicator\_id}`,
    - { headers: { Authorization: `Bearer ${process.env.TRADING\_ECONOMICS\_API\_KEY}` } }
    - );
    - const timeseries = await response.json();
    - await supabase.from('raw\_timeseries').insert(timeseries.map(t => ({
    - country\_id: country.country\_id,
    - indicator\_id: indicator.indicator\_id,
    - date: t.date,
    - value: t.value,
    - ingestion\_date: new Date().toISOString()
    - })));
    - console.log(`Ingested timeseries for ${country.country\_id} - ${indicator.indicator\_id}`);
    - }
    - }
    - }
    - ingestTETimeseries().catch(console.error);
    - dashboard/scripts/test\_supabase\_connection.js:
    - import { createClient } from '@supabase/supabase-js';
    - import dotenv from 'dotenv';
    - dotenv.config();
    - const supabase = createClient(process.env.SUPABASE\_URL, process.env.SUPABASE\_SERVICE\_ROLE\_KEY);
    - async function testSupabaseConnection() {
    - try {
    - const { data, error } = await supabase.from('raw\_countries').select('country\_id').limit(1);
    - if (error) throw error;
    - console.log('Supabase connection successful:', data);
    - } catch (error) {
    - console.error('Supabase connection failed:', error);
    - }
    - }
    - testSupabaseConnection();
    - dashboard/scripts/transform\_to\_canonical.js:
    - import { createClient } from '@supabase/supabase-js';
    - import dotenv from 'dotenv';
    - dotenv.config();
    - const supabase = createClient(process.env.SUPABASE\_URL, process.env.SUPABASE\_SERVICE\_ROLE\_KEY);
    - async function transformToCanonical() {
    - const { data: rawCountries } = await supabase.from('raw\_countries').select('\*');
    - const canonicalCountries = rawCountries.map(c => ({
    - country\_code: c.country\_id,
    - country\_name: c.country\_name,
    - ingestion\_date: c.ingestion\_date
    - }));
    - await supabase.from('canonical\_countries').insert(canonicalCountries);
    - const { data: rawIndicators } = await supabase.from('raw\_indicators').select('\*');
    - const canonicalIndicators = rawIndicators.map(i => ({
    - indicator\_name: i.indicator\_name,
    - ingestion\_date: i.ingestion\_date
    - }));
    - await supabase.from('canonical\_indicators').insert(canonicalIndicators);
    - const { data: rawTimeseries } = await supabase.from('raw\_timeseries').select('\*');
    - const canonicalTimeseries = rawTimeseries.map(t => ({
    - country\_code: t.country\_id,
    - indicator\_id: t.indicator\_id,
    - date: t.date,
    - value: t.value,
    - ingestion\_date: t.ingestion\_date
    - }));
    - await supabase.from('canonical\_timeseries').insert(canonicalTimeseries);
    - console.log(`Transformed ${canonicalCountries.length} countries, ${canonicalIndicators.length} indicators, ${canonicalTimeseries.length} timeseries`);
    - }
    - transformToCanonical().catch(console.error);
    - dashboard/scripts/validate\_data.js:
    - import { createClient } from '@supabase/supabase-js';
    - import dotenv from 'dotenv';
    - dotenv.config();
    - const supabase = createClient(process.env.SUPABASE\_URL, process.env.SUPABASE\_SERVICE\_ROLE\_KEY);
    - async function validateData() {
    - const { data: timeseries, error } = await supabase.from('canonical\_timeseries').select('\*');
    - if (error) throw error;
    - const invalidRecords = timeseries.filter(t => !t.value || !t.date);
    - if (invalidRecords.length > 0) {
    - console.error('Invalid records found:', invalidRecords);
    - await supabase.from('validation\_log').insert(
    - invalidRecords.map(r => ({
    - table\_name: 'canonical\_timeseries',
    - record\_id: r.timeseries\_id,
    - error\_message: 'Missing value or date',
    - validation\_date: new Date().toISOString()
    - }))
    - );
    - } else {
    - console.log('Data validation passed');
    - }
    - }
    - validateData().catch(console.error);
  + **Verification & Test Criteria**:
    - Ingestion scripts run: cd dashboard && node scripts/ingest\_te\_indicators.js, node scripts/ingest\_te\_timeseries.js.
    - Transformation script run: node scripts/transform\_to\_canonical.js.
    - Validation script run: node scripts/validate\_data.js.
    - Test: psql -c "SELECT COUNT(\*) FROM canonical\_countries;" returns non-zero rows; psql -c "SELECT \* FROM validation\_log;" shows no errors.

**Milestone 4: Phase 3, Stage 1 - Forecasting Pipeline**

* **✅ Status**: Completed
* **Deliverables Snapshot**:
  + **File Structure**:
  + E:\Users\Public\globalpulse\forecasting\scripts
  + ├── fetch\_data.py
  + ├── preprocess\_data.py
  + ├── test\_fetch\_preprocess.py
  + ├── test\_edge\_cases.py
  + ├── forecast\_data.py
  + ├── test\_forecast\_data.py
  + ├── test\_forecast\_edge\_cases.py
  + ├── visualize\_forecasts.py
  + **Files**:
    - forecasting/scripts/fetch\_data.py:
    - import pandas as pd
    - from supabase import create\_client
    - from dotenv import load\_dotenv
    - import os
    - load\_dotenv()
    - supabase = create\_client(os.getenv("SUPABASE\_URL"), os.getenv("SUPABASE\_SERVICE\_ROLE\_KEY"))
    - def fetch\_validated\_data():
    - countries = pd.DataFrame(supabase.from\_('canonical\_countries').select('\*').execute().data)
    - indicators = pd.DataFrame(supabase.from\_('canonical\_indicators').select('\*').execute().data)
    - timeseries = pd.DataFrame(supabase.from\_('canonical\_timeseries').select('\*').execute().data)
    - return countries, indicators, timeseries
    - forecasting/scripts/preprocess\_data.py:
    - import pandas as pd
    - def preprocess\_data(countries, indicators, timeseries):
    - timeseries['date'] = pd.to\_datetime(timeseries['date'])
    - timeseries = timeseries.sort\_values(['country\_code', 'indicator\_id', 'date'])
    - timeseries = timeseries.groupby(['country\_code', 'indicator\_id']).filter(lambda x: len(x) >= 12)
    - return countries, indicators, timeseries
    - forecasting/scripts/test\_fetch\_preprocess.py:
    - import pandas as pd
    - from fetch\_data import fetch\_validated\_data
    - from preprocess\_data import preprocess\_data
    - def test\_fetch\_preprocess():
    - countries, indicators, timeseries = fetch\_validated\_data()
    - assert not countries.empty, "No countries fetched"
    - assert not indicators.empty, "No indicators fetched"
    - assert not timeseries.empty, "No timeseries fetched"
    - countries, indicators, timeseries = preprocess\_data(countries, indicators, timeseries)
    - assert all(timeseries.groupby(['country\_code', 'indicator\_id']).size() >= 12), "Timeseries with <12 records found"
    - print("Fetch and preprocess tests passed")
    - if \_\_name\_\_ == "\_\_main\_\_":
    - test\_fetch\_preprocess()
    - forecasting/scripts/test\_edge\_cases.py:
    - import pandas as pd
    - from fetch\_data import fetch\_validated\_data
    - def test\_edge\_cases():
    - countries, indicators, timeseries = fetch\_validated\_data()
    - assert not countries['country\_code'].isnull().any(), "Null country\_code found"
    - assert not indicators['indicator\_id'].isnull().any(), "Null indicator\_id found"
    - assert not timeseries['value'].isnull().any(), "Null value found"
    - assert not timeseries['date'].isnull().any(), "Null date found"
    - print("Edge case tests passed")
    - if \_\_name\_\_ == "\_\_main\_\_":
    - test\_edge\_cases()
    - forecasting/scripts/forecast\_data.py:
    - import pandas as pd
    - from supabase import create\_client
    - from statsmodels.tsa.arima.model import ARIMA
    - from datetime import datetime
    - from dotenv import load\_dotenv
    - import os
    - load\_dotenv()
    - supabase = create\_client(os.getenv("SUPABASE\_URL"), os.getenv("SUPABASE\_SERVICE\_ROLE\_KEY"))
    - def forecast\_data(countries, indicators, timeseries):
    - forecasts = []
    - for \_, country in countries.iterrows():
    - for \_, indicator in indicators.iterrows():
    - data = timeseries[(timeseries['country\_code'] == country['country\_code']) &
    - (timeseries['indicator\_id'] == indicator['indicator\_id'])]
    - if len(data) >= 12:
    - model = ARIMA(data['value'], order=(1,1,1)).fit()
    - forecast = model.forecast(steps=3)
    - forecasts.append({
    - 'country\_code': country['country\_code'],
    - 'indicator\_id': indicator['indicator\_id'],
    - 'forecast\_date': datetime.now().isoformat(),
    - 'forecast\_values': forecast.tolist()
    - })
    - if forecasts:
    - supabase.from\_('forecast\_results').insert(forecasts).execute()
    - return forecasts
    - forecasting/scripts/test\_forecast\_data.py:
    - import pandas as pd
    - from fetch\_data import fetch\_validated\_data
    - from forecast\_data import forecast\_data
    - def test\_forecast\_data():
    - countries, indicators, timeseries = fetch\_validated\_data()
    - forecasts = forecast\_data(countries, indicators, timeseries)
    - assert len(forecasts) > 0, "No forecasts generated"
    - assert all('forecast\_values' in f for f in forecasts), "Missing forecast\_values"
    - print("Forecast data tests passed")
    - if \_\_name\_\_ == "\_\_main\_\_":
    - test\_forecast\_data()
    - forecasting/scripts/test\_forecast\_edge\_cases.py:
    - import pandas as pd
    - from fetch\_data import fetch\_validated\_data
    - def test\_forecast\_edge\_cases():
    - countries, indicators, timeseries = fetch\_validated\_data()
    - empty\_timeseries = timeseries.head(0)
    - forecasts = forecast\_data(countries, indicators, empty\_timeseries)
    - assert len(forecasts) == 0, "Forecasts generated for empty data"
    - print("Forecast edge case tests passed")
    - if \_\_name\_\_ == "\_\_main\_\_":
    - test\_forecast\_edge\_cases()
    - forecasting/scripts/visualize\_forecasts.py:
    - import pandas as pd
    - import matplotlib.pyplot as plt
    - from fetch\_data import fetch\_validated\_data
    - def visualize\_forecasts():
    - countries, indicators, timeseries = fetch\_validated\_data()
    - for \_, country in countries.iterrows():
    - for \_, indicator in indicators.iterrows():
    - data = timeseries[(timeseries['country\_code'] == country['country\_code']) &
    - (timeseries['indicator\_id'] == indicator['indicator\_id'])]
    - if not data.empty:
    - plt.plot(data['date'], data['value'], label=f"{country['country\_name']} - {indicator['indicator\_name']}")
    - plt.title(f"{country['country\_name']} - {indicator['indicator\_name']}")
    - plt.xlabel('Date')
    - plt.ylabel('Value')
    - plt.legend()
    - plt.show()
    - if \_\_name\_\_ == "\_\_main\_\_":
    - visualize\_forecasts()
  + **Verification & Test Criteria**:
    - Fetch and preprocess: py forecasting/scripts/test\_fetch\_preprocess.py outputs Fetch and preprocess tests passed.
    - Edge cases: py forecasting/scripts/test\_edge\_cases.py outputs Edge case tests passed.
    - Forecasting: py forecasting/scripts/forecast\_data.py stores forecasts in forecast\_results.
    - Test: psql -c "SELECT COUNT(\*) FROM forecast\_results;" returns non-zero rows.

**Milestone 5: Phase 3, Stage 2 - Taxonomy Mapping Logic and Implementation**

* **✅ Status**: Completed
* **Deliverables Snapshot**:
  + **File Structure**:
  + E:\Users\Public\globalpulse\forecasting\scripts
  + ├── map\_taxonomy.py
  + ├── test\_taxonomy\_mapping.py
  + ├── visualize\_taxonomy.py
  + E:\Users\Public\globalpulse\db\queries
  + ├── verify\_taxonomy\_log.sql
  + ├── verify\_taxonomy\_integrity.sql
  + ├── diagnose\_taxonomy\_mappings.sql
  + **Files**:
    - forecasting/scripts/map\_taxonomy.py:
    - import pandas as pd
    - from supabase import create\_client, Client
    - from datetime import datetime
    - from dotenv import load\_dotenv
    - import os
    - from fetch\_data import fetch\_validated\_data
    - load\_dotenv()
    - supabase\_url = os.getenv("SUPABASE\_URL")
    - supabase\_key = os.getenv("SUPABASE\_SERVICE\_ROLE\_KEY")
    - supabase = create\_client(supabase\_url, supabase\_key)
    - def map\_taxonomy(countries, indicators):
    - try:
    - start\_time = datetime.now()
    - mappings = []
    - categories\_response = supabase.table("categories").select("category\_id, category\_name").execute()
    - categories = pd.DataFrame(categories\_response.data)
    - category\_map = categories.set\_index('category\_name')['category\_id'].to\_dict()
    - category\_keywords = {
    - 'Growth': ['gdp', 'industrial production', 'economic growth'],
    - 'Prices': ['cpi', 'ppi', 'inflation', 'price index'],
    - 'Labor': ['unemployment', 'employment', 'labor force'],
    - 'Trade': ['trade balance', 'export', 'import'],
    - 'Sentiment': ['consumer confidence', 'business confidence', 'sentiment']
    - }
    - for \_, indicator in indicators.iterrows():
    - indicator\_name = indicator['indicator\_name'].lower()
    - indicator\_id = int(indicator['indicator\_id'])
    - matched\_category\_name = None
    - provenance\_rule = None
    - for category\_name, keywords in category\_keywords.items():
    - for keyword in keywords:
    - if keyword in indicator\_name:
    - matched\_category\_name = category\_name
    - provenance\_rule = f"Matched keyword: {keyword}"
    - break
    - if matched\_category\_name:
    - break
    - if matched\_category\_name:
    - category\_id = category\_map.get(matched\_category\_name)
    - for \_, country in countries.iterrows():
    - country\_code = country['country\_code']
    - mappings.append({
    - 'country\_code': country\_code,
    - 'indicator\_id': indicator\_id,
    - 'category\_id': category\_id,
    - 'rank': 1,
    - 'is\_primary': True,
    - 'fallback\_reason': None,
    - 'coverage\_ratio': 1.0,
    - 'provenance': {'source': 'keyword\_mapping', 'rule': provenance\_rule}
    - })
    - if mappings:
    - supabase.table("taxonomy\_mapping").delete().not\_.is\_('country\_code', 'null').execute()
    - supabase.table("taxonomy\_mapping").insert(mappings).execute()
    - print(f"Stored {len(mappings)} taxonomy mappings")
    - end\_time = datetime.now()
    - execution\_time\_ms = int((end\_time - start\_time).total\_seconds() \* 1000)
    - log\_entry = {
    - "endpoint": "map\_taxonomy",
    - "status": "success",
    - "records\_processed": len(mappings),
    - "error\_message": None,
    - "started\_at": start\_time.isoformat(),
    - "completed\_at": end\_time.isoformat(),
    - "execution\_time\_ms": execution\_time\_ms
    - }
    - supabase.table("ingestion\_log").insert(log\_entry).execute()
    - return mappings
    - except Exception as e:
    - print(f"Error mapping taxonomy: {str(e)}")
    - end\_time = datetime.now()
    - execution\_time\_ms = int((end\_time - start\_time).total\_seconds() \* 1000)
    - log\_entry = {
    - "endpoint": "map\_taxonomy",
    - "status": "failed",
    - "records\_processed": 0,
    - "error\_message": str(e),
    - "started\_at": start\_time.isoformat(),
    - "completed\_at": end\_time.isoformat(),
    - "execution\_time\_ms": execution\_time\_ms
    - }
    - supabase.table("ingestion\_log").insert(log\_entry).execute()
    - raise
    - if \_\_name\_\_ == "\_\_main\_\_":
    - countries, indicators, \_ = fetch\_validated\_data()
    - mappings = map\_taxonomy(countries, indicators)
    - forecasting/scripts/test\_taxonomy\_mapping.py:
    - import pandas as pd
    - from supabase import create\_client, Client
    - from datetime import datetime
    - from dotenv import load\_dotenv
    - import os
    - from fetch\_data import fetch\_validated\_data
    - import map\_taxonomy
    - def test\_taxonomy\_mapping():
    - try:
    - load\_dotenv()
    - supabase\_url = os.getenv("SUPABASE\_URL")
    - supabase\_key = os.getenv("SUPABASE\_SERVICE\_ROLE\_KEY")
    - supabase = create\_client(supabase\_url, supabase\_key)
    - countries, indicators, timeseries = fetch\_validated\_data()
    - mappings = map\_taxonomy.map\_taxonomy(countries, indicators)
    - assert len(mappings) > 0, "No mappings were generated."
    - mappings\_df = pd.DataFrame(mappings)
    - critical\_fields = ['country\_code', 'indicator\_id', 'category\_id', 'rank', 'is\_primary']
    - for field in critical\_fields:
    - assert not mappings\_df[field].isnull().any(), f"Null values found in {field}"
    - duplicates = mappings\_df.duplicated(subset=['country\_code', 'indicator\_id', 'category\_id']).sum()
    - assert duplicates == 0, f"Found {duplicates} duplicate mappings"
    - valid\_countries = set(countries['country\_code'])
    - assert all(mappings\_df['country\_code'].isin(valid\_countries)), "Invalid country\_code in mappings"
    - valid\_indicators = set(indicators['indicator\_id'])
    - assert all(mappings\_df['indicator\_id'].isin(valid\_indicators)), "Invalid indicator\_id in mappings"
    - valid\_categories = set(supabase.table("categories").select("category\_id").execute().data)
    - valid\_categories = {c['category\_id'] for c in valid\_categories}
    - assert all(mappings\_df['category\_id'].isin(valid\_categories)), "Invalid category\_id in mappings"
    - stored\_mappings = supabase.table("taxonomy\_mapping").select("mapping\_date").execute().data
    - mapping\_dates = pd.to\_datetime([m['mapping\_date'] for m in stored\_mappings])
    - assert all(mapping\_dates <= pd.Timestamp.now(tz='UTC')), "Future mapping\_date found"
    - assert all(mapping\_dates >= pd.Timestamp.now(tz='UTC') - pd.Timedelta(days=1)), "Old mapping\_date found"
    - stored\_results\_count = supabase.table("taxonomy\_mapping").select("count").execute().data[0]['count']
    - assert stored\_results\_count == len(mappings), f"Expected {len(mappings)} stored results, found {stored\_results\_count}"
    - print("Taxonomy mapping tests passed")
    - except Exception as e:
    - print(f"Taxonomy mapping test failed: {str(e)}")
    - raise
    - if \_\_name\_\_ == "\_\_main\_\_":
    - test\_taxonomy\_mapping()
    - forecasting/scripts/visualize\_taxonomy.py:
    - import pandas as pd
    - import matplotlib.pyplot as plt
    - from supabase import create\_client
    - from dotenv import load\_dotenv
    - import os
    - load\_dotenv()
    - supabase = create\_client(os.getenv("SUPABASE\_URL"), os.getenv("SUPABASE\_SERVICE\_ROLE\_KEY"))
    - def visualize\_taxonomy():
    - mappings = pd.DataFrame(supabase.table("taxonomy\_mapping").select("\*").execute().data)
    - categories = pd.DataFrame(supabase.table("categories").select("\*").execute().data)
    - merged = mappings.merge(categories, on="category\_id")
    - category\_counts = merged['category\_name'].value\_counts()
    - plt.bar(category\_counts.index, category\_counts.values)
    - plt.title("Taxonomy Mapping Distribution")
    - plt.xlabel("Category")
    - plt.ylabel("Number of Mappings")
    - plt.show()
    - if \_\_name\_\_ == "\_\_main\_\_":
    - visualize\_taxonomy()
    - db/queries/verify\_taxonomy\_log.sql:
    - SELECT \* FROM ingestion\_log WHERE endpoint = 'map\_taxonomy';
    - db/queries/verify\_taxonomy\_integrity.sql:
    - SELECT 'Invalid Country Code' AS test, m.country\_code
    - FROM taxonomy\_mapping m
    - WHERE NOT EXISTS (
    - SELECT 1 FROM canonical\_countries c WHERE c.country\_code = m.country\_code
    - )
    - UNION
    - SELECT 'Invalid Indicator ID' AS test, m.indicator\_id::text
    - FROM taxonomy\_mapping m
    - WHERE NOT EXISTS (
    - SELECT 1 FROM canonical\_indicators i WHERE i.indicator\_id = m.indicator\_id
    - )
    - UNION
    - SELECT 'Invalid Category ID' AS test, m.category\_id::text
    - FROM taxonomy\_mapping m
    - WHERE NOT EXISTS (
    - SELECT 1 FROM categories c WHERE c.category\_id = m.category\_id
    - );
    - db/queries/diagnose\_taxonomy\_mappings.sql:
    - SELECT 'Unmapped Indicators' AS test, i.indicator\_id, i.indicator\_name
    - FROM canonical\_indicators i
    - WHERE NOT EXISTS (
    - SELECT 1
    - FROM taxonomy\_mapping m
    - WHERE m.indicator\_id = i.indicator\_id
    - );
    - SELECT 'Mapping Distribution' AS test, c.category\_name, COUNT(m.mapping\_id) AS mapping\_count
    - FROM categories c
    - LEFT JOIN taxonomy\_mapping m ON c.category\_id = m.category\_id
    - GROUP BY c.category\_name;
  + **Verification & Test Criteria**:
    - Mapping script runs: py forecasting/scripts/map\_taxonomy.py.
    - Output: Stored 12 taxonomy mappings.
    - Tests pass: py forecasting/scripts/test\_taxonomy\_mapping.py outputs Taxonomy mapping tests passed.
    - Integrity checks: psql -f db/queries/verify\_taxonomy\_integrity.sql returns 0 rows.

**Milestone 6: Phase 4, Stage 1 - Frontend Dashboard Setup**

* **✅ Status**: Completed
* **Deliverables Snapshot**:
  + **File Structure**:
  + E:\Users\Public\globalpulse\frontend
  + ├── .env
  + ├── src
  + │ ├── lib
  + │ │ ├── supabase.ts
  + │ ├── components
  + │ │ ├── Dashboard.tsx
  + │ ├── \_\_tests\_\_
  + │ │ ├── supabase.test.ts
  + │ ├── App.tsx
  + │ ├── index.css
  + │ ├── setup.ts
  + ├── package.json
  + ├── tailwind.config.ts
  + ├── postcss.config.js
  + ├── vite.config.ts
  + ├── tsconfig.json
  + ├── tsconfig.app.json
  + ├── tsconfig.node.json
  + ├── node\_modules
  + **Files**:
    - frontend/.env:
    - VITE\_SUPABASE\_URL=https://jjizaincjbgigbylbbyi.supabase.co
    - VITE\_SUPABASE\_ANON\_KEY=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6ImpqaXphaW5jamJnaWdieWxiYnlpIiwicm9sZSI6ImFub24iLCJpYXQiOjE3NTYzMjA1NTMsImV4cCI6MjA3MTg5NjU1M30.SkkIilhr1CumUHUoQrwtjAPp6PrqTCQv-a5IQPgYNco
    - frontend/src/lib/supabase.ts:
    - import { createClient } from '@supabase/supabase-js';
    - const supabaseUrl = import.meta.env.VITE\_SUPABASE\_URL;
    - const supabaseKey = import.meta.env.VITE\_SUPABASE\_ANON\_KEY;
    - export const supabase = createClient(supabaseUrl, supabaseKey);
    - frontend/src/App.tsx:
    - import { BrowserRouter, Routes, Route } from 'react-router-dom';
    - import Dashboard from './components/Dashboard';
    - import './index.css';
    - function App() {
    - return (
    - <BrowserRouter>
    - <Routes>
    - <Route path="/" element={<Dashboard />} />
    - </Routes>
    - </BrowserRouter>
    - );
    - }
    - export default App;
    - frontend/src/components/Dashboard.tsx:
    - import { useEffect, useState } from 'react';
    - import { supabase } from '../lib/supabase';
    - function Dashboard() {
    - const [connected, setConnected] = useState(false);
    - useEffect(() => {
    - async function testConnection() {
    - try {
    - const { data, error } = await supabase.from('categories').select('category\_name').limit(1);
    - if (error) throw error;
    - setConnected(!!data);
    - } catch (error) {
    - console.error('Supabase connection failed:', error);
    - }
    - }
    - testConnection();
    - }, []);
    - return (
    - <div className="min-h-screen bg-gradient-to-b from-[#0F0F12] to-[#1A1A1D] text-white">
    - <nav className="fixed left-0 top-0 h-full w-16 bg-[#1A1A1D] flex flex-col items-center py-4">
    - <div className="mb-8 text-2xl">GP</div>
    - <a href="/" className="mb-4 text-[#7B29B8] hover:text-[#C129A0]"><i className="fas fa-home"></i></a>
    - </nav>
    - <main className="ml-16 p-8">
    - <h1 className="text-2xl font-mono uppercase tracking-wider text-[#7B29B8]">
    - GlobalPulse Dashboard
    - </h1>
    - <p className="mt-2 font-sans text-gray-300">
    - {connected ? 'Connected to Supabase' : 'Connecting to Supabase...'}
    - </p>
    - </main>
    - </div>
    - );
    - }
    - export default Dashboard;
    - frontend/src/index.css:
    - @tailwind base;
    - @tailwind components;
    - @tailwind utilities;
    - @import '@fortawesome/fontawesome-free/css/all.min.css';
    - :root {
    - --primary-bg: #0F0F12;
    - --secondary-bg: #1A1A1D;
    - --accent-ember: #7B29B8;
    - --accent-magenta: #C129A0;
    - --accent-cyan: #29B8B0;
    - --text-primary: #FFFFFF;
    - --text-secondary: #FFFFFF33;
    - }
    - body {
    - font-family: 'Inter', 'Open Sans', sans-serif;
    - }
    - h1, h2, h3 {
    - font-family: 'Space Mono', 'Fragment Mono', 'DM Mono', 'monospace';
    - text-transform: uppercase;
    - letter-spacing: 0.1em;
    - }
    - frontend/src/\_\_tests\_\_/supabase.test.ts:
    - import { describe, it, expect } from 'vitest';
    - import { supabase } from '../lib/supabase';
    - describe('Supabase Connection', () => {
    - it('should connect to Supabase and fetch categories', async () => {
    - const { data, error } = await supabase.from('categories').select('category\_name').limit(1);
    - expect(error).toBeNull();
    - expect(data).toBeDefined();
    - expect(Array.isArray(data)).toBe(true);
    - });
    - });
    - frontend/src/setup.ts:
    - import '@testing-library/jest-dom/vitest';
    - frontend/package.json:
    - {
    - "name": "globalpulse-frontend",
    - "private": true,
    - "version": "0.0.0",
    - "type": "module",
    - "scripts": {
    - "dev": "vite",
    - "build": "tsc && vite build",
    - "lint": "eslint . --ext ts,tsx --report-unused-disable-directives --max-warnings 0",
    - "preview": "vite preview",
    - "test": "vitest"
    - },
    - "dependencies": {
    - "@fortawesome/fontawesome-free": "^6.6.0",
    - "@supabase/supabase-js": "^2.45.4",
    - "react": "^18.3.1",
    - "react-dom": "^18.3.1",
    - "react-router-dom": "^6.26.2"
    - },
    - "devDependencies": {
    - "@testing-library/jest-dom": "^6.8.0",
    - "@testing-library/react": "^16.3.0",
    - "@types/react": "^18.3.3",
    - "@types/react-dom": "^18.3.0",
    - "@vitejs/plugin-react": "^4.3.1",
    - "autoprefixer": "^10.4.20",
    - "eslint": "^8.57.0",
    - "eslint-plugin-react": "^7.34.3",
    - "jsdom": "^24.1.3",
    - "postcss": "^8.4.41",
    - "tailwindcss": "^3.4.10",
    - "typescript": "^5.5.4",
    - "vite": "^5.4.2",
    - "vitest": "^1.6.1"
    - }
    - }
    - frontend/tailwind.config.ts:
    - import type { Config } from 'tailwindcss';
    - export default {
    - content: ['./index.html', './src/\*\*/\*.{js,ts,jsx,tsx}'],
    - theme: {
    - extend: {
    - colors: {
    - 'deep-void': '#0F0F12',
    - 'charcoal': '#1A1A1D',
    - 'quantum-ember': '#7B29B8',
    - 'radiant-magenta': '#C129A0',
    - 'stabilizer-cyan': '#29B8B0',
    - 'text-primary': '#FFFFFF',
    - 'text-secondary': '#FFFFFF33',
    - },
    - fontFamily: {
    - mono: ['Space Mono', 'Fragment Mono', 'DM Mono', 'monospace'],
    - sans: ['Inter', 'Open Sans', 'sans-serif'],
    - },
    - },
    - },
    - plugins: [],
    - } satisfies Config;
    - frontend/postcss.config.js:
    - export default {
    - plugins: {
    - 'tailwindcss/nesting': {},
    - tailwindcss: {},
    - autoprefixer: {},
    - },
    - };
    - frontend/vite.config.ts:
    - import { defineConfig } from 'vitest/config';
    - import react from '@vitejs/plugin-react';
    - export default defineConfig({
    - plugins: [react()],
    - test: {
    - globals: true,
    - environment: 'jsdom',
    - setupFiles: './src/setup.ts',
    - },
    - });
    - frontend/tsconfig.json:
    - {
    - "files": [],
    - "references": [
    - { "path": "./tsconfig.app.json" },
    - { "path": "./tsconfig.node.json" }
    - ]
    - }
    - frontend/tsconfig.app.json:
    - {
    - "compilerOptions": {
    - "tsBuildInfoFile": "./node\_modules/.tmp/tsconfig.app.tsbuildinfo",
    - "target": "ES2022",
    - "useDefineForClassFields": true,
    - "lib": ["ES2022", "DOM", "DOM.Iterable"],
    - "module": "ESNext",
    - "skipLibCheck": true,
    - "moduleResolution": "bundler",
    - "allowImportingTsExtensions": true,
    - "verbatimModuleSyntax": true,
    - "moduleDetection": "force",
    - "noEmit": true,
    - "jsx": "react-jsx",
    - "types": ["vitest/globals", "@testing-library/jest-dom"],
    - "strict": true,
    - "noUnusedLocals": true,
    - "noUnusedParameters": true,
    - "erasableSyntaxOnly": true,
    - "noFallthroughCasesInSwitch": true,
    - "noUncheckedSideEffectImports": true
    - },
    - "include": ["src"]
    - }
    - frontend/tsconfig.node.json:
    - {
    - "compilerOptions": {
    - "tsBuildInfoFile": "./node\_modules/.tmp/tsconfig.node.tsbuildinfo",
    - "target": "ES2023",
    - "lib": ["ES2023"],
    - "module": "ESNext",
    - "skipLibCheck": true,
    - "moduleResolution": "bundler",
    - "allowImportingTsExtensions": true,
    - "verbatimModuleSyntax": true,
    - "moduleDetection": "force",
    - "noEmit": true,
    - "strict": true,
    - "noUnusedLocals": true,
    - "noUnusedParameters": true,
    - "erasableSyntaxOnly": true,
    - "noFallthroughCasesInSwitch": true,
    - "noUncheckedSideEffectImports": true
    - },
    - "include": ["vite.config.ts"]
    - }
  + **Verification & Test Criteria**:
    - Frontend runs: cd frontend && npm run dev.
    - Output: Dashboard at http://localhost:5173 displays "GP", "GlobalPulse Dashboard", "Connected to Supabase".
    - Tests pass: npm run test outputs Test passed for Supabase connection.
    - UI theme: Dark gradient (#0F0F12 to #1A1A1D), purple accents (#7B29B8), Space Mono headings, Inter body text, Font Awesome icons.
    - Modularity: dir dashboard\scripts shows no changes.

**3. Design & Approach Change Log**

\*\*Change 1: Switch from Python to JavaScript for Supabase Connectivity\*\*

* **Original Design**: Used psycopg2 in test\_supabase\_connection.py for database connectivity.
* **Proposed Change**: Switch to @supabase/supabase-js in test\_supabase\_connection.js using HTTPS (port 443).
* **Rationale**: Python connection failed due to port 5432 timeout, likely due to firewall restrictions. JavaScript uses HTTPS, which is less restricted.
* **Decision**: Adopted.
* **Ripple Effect**: All subsequent Supabase interactions (ingestion, transformation) use @supabase/supabase-js in dashboard/scripts. Python reserved for future forecasting tasks.

1. **Change 2: Normalize Indicator Names in canonical\_indicators**
   * **Original Design**: Used API names (e.g., Consumer Price Index CPI) in canonical\_indicators.
   * **Proposed Change**: Normalize to lowercase (e.g., consumer price index cpi) to match raw\_te\_timeseries.
   * **Rationale**: Mismatch in naming caused potential mapping issues in time series transformation.
   * **Decision**: Adopted.
   * **Ripple Effect**: Ensures consistent filtering in transform\_to\_canonical.js, but requires re-running ingestion to align data.
2. **Change 3: Clear Tables Before Ingestion**
   * **Original Design**: Incremental ingestion without clearing tables.
   * **Proposed Change**: Clear raw\_te\_metadata, canonical\_indicators, canonical\_timeseries before ingestion.
   * **Rationale**: Stale data from previous runs caused inconsistent counts (e.g., 187 time series records).
   * **Decision**: Adopted.
   * **Ripple Effect**: Requires re-running ingest\_te\_timeseries.js and ingest\_te\_indicators.js after clearing.

* **Change 4: Taxonomy Mapping Logic Fix** (Phase 3, Stage 2)
  + **Original Design**: Hardcoded expected mapping counts; assumed all indicators map.
  + **Proposed Change**: Remove hardcoded counts, iterate keywords individually, delete existing mappings before insertion.
  + **Rationale**: Ensure scalability, prevent duplicates, handle unmapped indicators.
  + **Decision**: Adopted.
  + **Ripple Effect**: Updated map\_taxonomy.py, test\_taxonomy\_mapping.py; added diagnose\_taxonomy\_mappings.sql.
* **Change 5: Frontend Isolation** (Phase 4, Stage 1)
  + **Original Design**: Frontend setup in dashboard directory.
  + **Proposed Change**: Separate frontend directory with Vite, TypeScript, ESM.
  + **Rationale**: Ensure modularity, avoid conflicts with Node.js scripts, use modern tooling.
  + **Decision**: Adopted.
  + **Ripple Effect**: No changes to dashboard; new frontend structure with Vite, Tailwind CSS, Font Awesome.

**4. Noise Filtering**

* **Excluded**:
  + Initial buggy map\_taxonomy.py with undefined keyword variable.
  + Unresolved debugging chatter about mapping counts.
  + Previous incomplete frontend attempts with errors.
* **Included**:
  + Fixed map\_taxonomy.py with correct keyword iteration.
  + Confirmed JavaScript connection test.
  + Normalized indicator names in transform\_to\_canonical.js.
  + Table clearing SQL to resolve data inconsistencies.

**5. Last Verified State**

* **Most Recent Milestone**: Phase 4, Stage 1 - Frontend Dashboard Setup
* **Full Project File Tree**:
* E:\Users\Public\globalpulse
* ├── .env
* ├── dashboard
* │ ├── scripts
* │ │ ├── ingest\_te\_indicators.js
* │ │ ├── ingest\_te\_timeseries.js
* │ │ ├── test\_supabase\_connection.js
* │ │ ├── transform\_to\_canonical.js
* │ │ ├── validate\_data.js
* │ ├── package.json
* │ ├── node\_modules
* ├── forecasting
* │ ├── .env
* │ ├── scripts
* │ │ ├── fetch\_data.py
* │ │ ├── preprocess\_data.py
* │ │ ├── test\_fetch\_preprocess.py
* │ │ ├── test\_edge\_cases.py
* │ │ ├── forecast\_data.py
* │ │ ├── test\_forecast\_data.py
* │ │ ├── test\_forecast\_edge\_cases.py
* │ │ ├── visualize\_forecasts.py
* │ │ ├── map\_taxonomy.py
* │ │ ├── test\_taxonomy\_mapping.py
* │ │ ├── visualize\_taxonomy.py
* │ ├── requirements.txt
* │ ├── venv
* ├── frontend
* │ ├── .env
* │ ├── src
* │ │ ├── lib
* │ │ │ ├── supabase.ts
* │ │ ├── components
* │ │ │ ├── Dashboard.tsx
* │ │ ├── \_\_tests\_\_
* │ │ │ ├── supabase.test.ts
* │ │ ├── App.tsx
* │ │ ├── index.css
* │ │ ├── setup.ts
* │ ├── package.json
* │ ├── tailwind.config.ts
* │ ├── postcss.config.js
* │ ├── vite.config.ts
* │ ├── tsconfig.json
* │ ├── tsconfig.app.json
* │ ├── tsconfig.node.json
* │ ├── node\_modules
* ├── db
* │ ├── migrations
* │ │ ├── 001\_create\_tables.sql
* │ │ ├── 002\_create\_rls\_policies.sql
* │ │ ├── 003\_clear\_tables.sql
* │ │ ├── 004\_create\_validation\_functions.sql
* │ │ ├── 005\_create\_validation\_log.sql
* │ │ ├── 006\_create\_forecast\_results.sql
* │ │ ├── 007\_create\_taxonomy\_tables.sql
* │ │ ├── 008\_seed\_categories.sql
* │ ├── queries
* │ │ ├── check\_duplicates.sql
* │ │ ├── diagnose\_validation\_failure.sql
* │ │ ├── diagnose\_timeseries\_count.sql
* │ │ ├── test\_validation\_edge\_cases.sql
* │ │ ├── verify\_data.sql
* │ │ ├── verify\_validation.sql
* │ │ ├── verify\_preprocess\_log.sql
* │ │ ├── verify\_forecast\_results.sql
* │ │ ├── verify\_forecast\_log.sql
* │ │ ├── verify\_taxonomy\_log.sql
* │ │ ├── verify\_taxonomy\_integrity.sql
* │ │ ├── diagnose\_taxonomy\_mappings.sql
* **Current Codebase Snapshot**: See Milestone 6 files.
* **Current DB Schemas**:
  + raw\_countries, raw\_indicators, raw\_timeseries
  + canonical\_countries, canonical\_indicators, canonical\_timeseries
  + categories, taxonomy\_mapping, forecast\_results
  + validation\_log, ingestion\_log
* **Current Automation Workflows**: None (n8n/Zapier not implemented).
* **Admin Dashboard State**: React dashboard with Supabase connection, displaying "GP", "GlobalPulse Dashboard", "Connected to Supabase".
* **Outstanding Issues**:
  + Taxonomy mappings: Only 12 mappings stored (expected ~16 for 4 countries \* 4 indicators).
  + No data visualizations in dashboard yet.

**6. Consistency & Risk Analysis**

* **Contradictions**:
  + None; schemas and scripts align.
* **Risks**:
  + **Incomplete Taxonomy Mappings**: Only 12 mappings stored, potentially due to missing keywords.
    - **Mitigation**: Run db/queries/diagnose\_taxonomy\_mappings.sql; update category\_keywords in map\_taxonomy.py.
  + **Supabase Anon Key Security**: Public anon key in frontend .env requires restricted permissions.
    - **Mitigation**: Verify RLS policies in Supabase dashboard; limit anon key to select on categories, taxonomy\_mapping, forecast\_results.
  + **Dependency Version Drift**: Python/Node.js dependencies may become outdated.
    - **Mitigation**: Pin versions in requirements.txt and package.json.
* **Dependencies Skipped**:
  + None; all required tables and scripts delivered.

**7. Next Step Readiness**

* **Next Phase/Stage/Step**: Phase 4, Stage 2 - Dashboard Data Integration and Visualization
* **Dependencies**:
  + **Delivered**:
    - Database tables: categories, taxonomy\_mapping, forecast\_results.
    - Frontend setup: React with TypeScript, Supabase client, Tailwind CSS, Font Awesome.
    - Data pipeline: Ingestion, transformation, forecasting, taxonomy mapping.
  + **Required for Next Stage**:
    - Fetch taxonomy\_mapping and forecast\_results from Supabase.
    - Visualize data using charts (e.g., Chart.js).
    - Implement routing for views (e.g., by country or category).
* **Handoff Context**:
  + **Deliverables to Consume**:
    - frontend/src/lib/supabase.ts for data fetching.
    - taxonomy\_mapping, forecast\_results tables for visualization.
  + **Verified Inputs/Outputs**:
    - taxonomy\_mapping: 12 mappings stored, verified by test\_taxonomy\_mapping.py.
    - Dashboard: Displays "Connected to Supabase" at http://localhost:5173.
  + **Assumptions**:
    - **Safe**: Supabase anon key has select permissions; UI theme correct.
    - **Risky**: All indicators mapped; may need keyword updates.
* **Setup Commands**:
* cd E:\Users\Public\globalpulse\frontend
* npm install
* npm run dev