# E-Commerce Project Implementation Guide (Supabase + Next.js + Vercel)

This document describes the step-by-step implementation of the E-commerce project using Supabase as backend, Next.js as frontend, Cursor IDE for development, GitHub for version control, and Vercel for deployment.

## Phase 1 — Planning & Analysis

Deliverables:  
• Define the scope, MVP features, and data model (users, stores, products, variants, orders, requests).  
• Deployment plan (Cursor → GitHub → Vercel; Supabase project and DB).  
• Authentication via Supabase Auth.  
• Currency default: XAF (CFA).

Core database schema (SQL to run in Supabase):

CREATE TABLE profiles (...);  
CREATE TABLE stores (...);  
CREATE TABLE products (...);  
CREATE TABLE product\_variants (...);  
CREATE TABLE orders (...);  
CREATE TABLE order\_items (...);  
CREATE TABLE product\_requests (...);

## Phase 2 — Frontend Prototype (Next.js + Tailwind)

Steps:  
1. Create a Next.js project.  
2. Configure Tailwind and Supabase client.  
3. Create pages: Home (product list), Product Detail, Cart.  
4. Implement authentication flow with Supabase Auth.  
5. Connect Supabase database to frontend for fetching products.

Example: Supabase client setup (lib/supabaseClient.ts)

import { createClient } from '@supabase/supabase-js'  
const supabase = createClient(process.env.NEXT\_PUBLIC\_SUPABASE\_URL!, process.env.NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY!)  
export { supabase }

Example: Product listing (pages/index.tsx)

export const getServerSideProps = async () => {  
 const { data: products } = await supabase.from('products').select('\*').eq('active', true)  
 return { props: { products } }  
}

## Phase 3 — Backend & API Integration

Steps:  
1. Configure Supabase project and environment variables.  
2. Create SQL schema and seed data.  
3. Implement API routes in Next.js (pages/api/).  
4. Add authentication and RLS policies.  
5. Deploy to Vercel with proper env vars.

Example API route (pages/api/create-order.ts):

import { createClient } from '@supabase/supabase-js'  
const supabase = createClient(process.env.NEXT\_PUBLIC\_SUPABASE\_URL!, process.env.SUPABASE\_SERVICE\_ROLE\_KEY!)  
  
export default async function handler(req, res) {  
 if (req.method !== 'POST') return res.status(405).end()  
 const { userId, storeId, cart, shipping } = req.body  
 let total = cart.reduce((sum, i) => sum + i.unit\_price \* i.quantity, 0)  
 const { data, error } = await supabase.from('orders').insert([{ user\_id: userId, store\_id: storeId, total, shipping\_info: shipping }]).select('\*').single()  
 if (error) return res.status(500).json({ error })  
 res.status(200).json({ orderId: data.id })  
}

## Deployment Workflow

1. Push code to GitHub.  
2. Connect repository to Vercel.  
3. Add environment variables:  
 - NEXT\_PUBLIC\_SUPABASE\_URL  
 - NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY  
 - SUPABASE\_SERVICE\_ROLE\_KEY  
4. Deploy main branch to production.

## Testing

Run `npm run dev` to test locally. Use Supabase dashboard to insert test data and verify product list, cart, and order creation.