

CouponSwap

Tech Stack & Implementation Guide

Complete Step-by-Step Workflow with AI Prompts

Document Type:	Implementation Handbook
Version:	1.0
Date:	January 31, 2026
Target Audience:	Beginner Developers (2nd Year CS Students)
Estimated Time:	10-12 weeks to MVP

Table of Contents

- 1. Tech Stack Overview
- 2. Prerequisites & Tool Installation
- 3. Phase 0: Environment Setup (Manual)
- 4. Phase 1: Database Setup (Supabase)
- 5. Phase 2: Frontend Development (Gemini AI Prompts)
- 6. Phase 3: Integration (VS Code + GitHub)
- 7. Phase 4: Deployment (Vercel)
- 8. Phase 5: AI Features (Python ML)
- 9. Troubleshooting Guide
- 10. Appendix: Complete Prompt Library

1. Tech Stack Overview

This project uses a modern, beginner-friendly tech stack optimized for rapid development and free deployment. Every tool has been selected for its ease of use, extensive documentation, and suitability for student projects.

Category	Tool/Technology	Version	Purpose	Cost
Frontend	Next.js	14.x	React framework with App Router	Free
	React	18.x	UI library	Free
	TypeScript	5.x	Type-safe JavaScript	Free
	Tailwind CSS	3.x	Utility-first CSS framework	Free
	Lucide React	Latest	Icon library	Free
Backend	Supabase	Cloud	PostgreSQL, Auth, Realtime, Storage	Free tier
	PostgreSQL	15.x	Relational database	Included
Hosting	Vercel	Cloud	Frontend deployment & CDN	Free tier
AI/ML	Google Gemini API	1.5/2.0	AI code generation & OCR	Free tier
	Python	3.10+	ML recommendation engine	Free
	Scikit-learn	Latest	Machine learning library	Free
Dev Tools	VS Code	Latest	Code editor	Free
	Git	2.x	Version control	Free
	GitHub	Cloud	Code hosting	Free
	Node.js	20 LTS	JavaScript runtime	Free

Total Monthly Cost for 10K Users: \$0 (Using free tiers)

2. Prerequisites & Tool Installation

2.1 Required Accounts (Sign up first)

GitHub: <https://github.com/signup>

Purpose: Code hosting and version control

Vercel: <https://vercel.com/signup>

Purpose: Frontend deployment (sign in with GitHub)

Supabase: <https://supabase.com/dashboard>

Purpose: Backend database and auth

Google AI Studio: <https://aistudio.google.com>

Purpose: Access to Gemini AI (optional but recommended)

2.2 Software Installation (Step-by-Step)

STEP 1: Install Node.js

1. Go to <https://nodejs.org>
2. Download the "LTS" version (Long Term Support)
3. Run installer, click "Next" for all options
4. Verify installation:
 - Open terminal/command prompt
 - Type: node --version
 - Should show: v20.x.x or similar

STEP 2: Install Git

1. Go to <https://git-scm.com/downloads>
2. Download for your OS (Windows/Mac/Linux)
3. Run installer with default settings
4. Verify: git --version

STEP 3: Install VS Code

1. Go to <https://code.visualstudio.com>
2. Download and install
3. Recommended extensions (install from Extensions panel):
 - ESLint
 - Prettier
 - Tailwind CSS IntelliSense
 - GitLens

STEP 4: Configure Git (First time only)

Open terminal in VS Code (Ctrl+J or Cmd+J) and run:

```
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```

3. Phase 0: Environment Setup (Manual Steps)

3.1 Create Next.js Project

1. Open terminal (Command Prompt on Windows, Terminal on Mac)
2. Navigate to where you want your project:
cd Desktop (or any folder you prefer)
3. Run the create command:
npx create-next-app@latest coupon-swap
4. Answer the prompts:
 - ✓ Would you like to use TypeScript? → Yes
 - ✓ Would you like to use ESLint? → Yes
 - ✓ Would you like to use Tailwind CSS? → Yes
 - ✓ Would you like to use `src/` directory? → No
 - ✓ Would you like to use App Router? → Yes
 - ✓ Would you like to customize the default import alias? → No
5. Wait for installation (2-3 minutes)
6. Navigate into project:
cd coupon-swap
7. Open in VS Code:
code .

3.2 Install Additional Dependencies

In VS Code terminal (Ctrl+J to open), run:

```
npm install @supabase/supabase-js lucide-react clsx tailwind-merge
```

This installs:

- @supabase/supabase-js → Supabase client library
- lucide-react → Icon components
- clsx & tailwind-merge → Utility for conditional CSS classes

3.3 Test Local Development

1. In terminal, run:
npm run dev
2. You should see:
▲ Next.js 14.x.x
- Local: http://localhost:3000
- Ready in 2.3s
3. Open browser to http://localhost:3000
4. You should see the Next.js welcome page
5. Press Ctrl+C in terminal to stop the server

4. Phase 1: Database Setup (Supabase)

4.1 Create Supabase Project

1. Go to <https://supabase.com/dashboard>
2. Click 'New Project'
3. Fill in details:
 - Name: coupon-swap
 - Database Password: (Create a strong password and SAVE IT)
 - Region: Select closest to you (e.g., Mumbai for India)
4. Click 'Create new project'
5. Wait 2-3 minutes for provisioning

4.2 Get API Keys

1. In Supabase dashboard, click 'Project Settings' (gear icon)
2. Navigate to 'API' section in left sidebar
3. Find and copy these values:
 - **Project URL:** <https://xxxxx.supabase.co>
 - **anon/public key:** eyJhbGciOiJIUzI1NilsInR5cCl6IkpxVCJ9...
4. Keep this tab open - you'll need these values next

4.3 Create Environment File in VS Code

1. In VS Code, right-click in the file explorer (left sidebar)
2. Click "New File"
3. Name it exactly: .env.local
4. Paste this template:

```
NEXT_PUBLIC_SUPABASE_URL=your_project_url_here  
NEXT_PUBLIC_SUPABASE_ANON_KEY=your_anon_key_here
```

5. Replace the values with what you copied from Supabase
6. Save the file (Ctrl+S or Cmd+S)
7. IMPORTANT: This file should NEVER be committed to Git
(Next.js automatically ignores .env.local)

4.4 Run Database Schema Script

This is the most important step. Copy this ENTIRE SQL script and run it in Supabase SQL Editor to create all tables, triggers, and security policies.

```
-- =====
-- COUPONSWAP DATABASE SCHEMA
-- Run this in Supabase SQL Editor: Dashboard → SQL Editor → New Query
-- =====

-- 1. PROFILES TABLE (User Information)
create table public.profiles (
    id uuid references auth.users on delete cascade not null primary key,
    email text,
    phone text,
    kyc_id_number text,
    kyc_verified boolean default false,
    credits integer default 50 not null,
    created_at timestamp with time zone default timezone('utc'::text, now()) not null
);

-- 2. COUPONS TABLE (Marketplace Listings)
create table public.coupons (
    id uuid default uid_generate_v4() primary key,
    seller_id uuid references public.profiles(id) on delete cascade not null,
    title text not null,
    description text,
    code text not null,
    price_credits integer not null check (price_credits > 0),
    category text,
    expiry_date date,
    image_url text,
    is_sold boolean default false,
    buyer_id uuid references public.profiles(id) on delete set null,
    created_at timestamp with time zone default timezone('utc'::text, now()) not null
);

-- 3. TRANSACTIONS TABLE (Credit Transfer History)
create table public.transactions (
    id uuid default uid_generate_v4() primary key,
    buyer_id uuid references public.profiles(id) on delete cascade,
    seller_id uuid references public.profiles(id) on delete cascade,
    amount_credits integer not null,
    coupon_id uuid references public.coupons(id) on delete set null,
    created_at timestamp with time zone default timezone('utc'::text, now()) not null
);

-- 4. MESSAGES TABLE (Chat System)
create table public.messages (
    id uuid default uid_generate_v4() primary key,
    sender_id uuid references public.profiles(id) on delete cascade not null,
    receiver_id uuid references public.profiles(id) on delete cascade not null,
    content text not null,
    created_at timestamp with time zone default timezone('utc'::text, now()) not null
);

-- =====
-- INDEXES (Performance Optimization)
-- =====
create index idx_coupons_is_sold on public.coupons(is_sold);
create index idx_coupons_category on public.coupons(category);
create index idx_coupons_seller on public.coupons(seller_id);
create index idx_messages_sender on public.messages(sender_id);
create index idx_messages_receiver on public.messages(receiver_id);

-- =====
-- TRIGGER: Auto-create profile on user signup
-- =====
create or replace function public.handle_new_user()
returns trigger
language plpgsql
security definer
```

```

as $$

begin
    insert into public.profiles (id, email, credits)
    values (new.id, new.email, 50);
    return new;
end;
$$;

create trigger on_auth_user_created
after insert on auth.users
for each row execute procedure public.handle_new_user();

-- =====
-- ROW-LEVEL SECURITY (RLS) POLICIES
-- =====

-- Enable RLS on all tables
alter table public.profiles enable row level security;
alter table public.coupons enable row level security;
alter table public.transactions enable row level security;
alter table public.messages enable row level security;

-- PROFILES: Users can view and edit their own profile
create policy "Users can view own profile"
on public.profiles for select
using (auth.uid() = id);

create policy "Users can update own profile"
on public.profiles for update
using (auth.uid() = id);

-- COUPONS: Anyone can view unsold coupons
create policy "Anyone can view unsold coupons"
on public.coupons for select
using (is_sold = false OR auth.uid() = seller_id OR auth.uid() = buyer_id);

-- COUPONS: Users can insert their own coupons
create policy "Users can insert own coupons"
on public.coupons for insert
with check (auth.uid() = seller_id);

-- COUPONS: Sellers can update their own unsold coupons
create policy "Sellers can update own coupons"
on public.coupons for update
using (auth.uid() = seller_id);

-- MESSAGES: Users can view messages they sent or received
create policy "Users can view own messages"
on public.messages for select
using (auth.uid() = sender_id OR auth.uid() = receiver_id);

-- MESSAGES: Users can send messages
create policy "Users can send messages"
on public.messages for insert
with check (auth.uid() = sender_id);

-- TRANSACTIONS: Users can view their own transactions
create policy "Users can view own transactions"
on public.transactions for select
using (auth.uid() = buyer_id OR auth.uid() = seller_id);

-- =====
-- SUCCESS MESSAGE
-- =====
-- If you see this without errors, your database is ready!

```

How to run this script:

1. Copy the entire SQL script above
2. In Supabase dashboard, click 'SQL Editor' in left sidebar

3. Click 'New query'
4. Paste the script
5. Click 'Run' button (bottom right)
6. Wait for 'Success. No rows returned' message

5. Phase 2: Frontend Development (AI-Powered)

This is where we use Google Gemini or Claude AI to generate code. Copy these prompts ONE BY ONE into your AI tool. After each prompt, paste the generated code into the appropriate file in VS Code.

5.1 Initialize Supabase Client

PROMPT #1 (Copy this into Gemini/Claude):

```
I am building a Next.js 14 App Router application with Supabase and TypeScript. Task 1: Create a file lib/supabase.ts This file should initialize the Supabase client using: • process.env.NEXT_PUBLIC_SUPABASE_URL • process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY Task 2: Create a file types/index.ts Define TypeScript interfaces for: • Profile (id, email, credits, kyc_verified) • Coupon (id, title, description, code, price_credits, category, expiry_date, image_url, is_sold, seller_id, buyer_id) • Message (id, sender_id, receiver_id, content, created_at) Provide complete code for both files.
```

What to do with the output:

1. Create folder 'lib' in your project root (if not exists)
2. Create file lib/supabase.ts and paste the generated code
3. Create folder 'types' in your project root
4. Create file types/index.ts and paste the generated code
5. Save both files (Ctrl+S)

5.2 Create Navigation Bar

PROMPT #2:

```
Create a responsive navigation bar component at components/Navbar.tsx Requirements: • Logo/title "CouponSwap" on the left • Navigation links: Home (/), Sell Coupon (/sell) • Right side: If user is logged in, show "Credits: [balance]" and "Logout" button. If not logged in, show "Login" button. • Use Tailwind CSS for styling (modern gradient background) • Use lucide-react for icons (Menu, X for mobile hamburger) • Mobile responsive (hamburger menu for < 768px) • Get user data from Supabase auth: const { data: { user } } = await supabase.auth.getUser() • Fetch credit balance from profiles table Provide complete code with all imports.
```

What to do:

1. Create folder 'components'
2. Create file components/Navbar.tsx
3. Paste generated code

4. Update app/layout.tsx to import and use

5.3 Login/Signup Page

PROMPT #3:

```
Create a login/signup page at app/login/page.tsx Requirements: • Centered card layout with tabs: "Sign In" and "Sign Up" • Sign In form: Email, Password, Submit button • Sign Up form: Email, Password, Confirm Password, KYC ID Number (optional), Submit button • Use Supabase Auth: - signIn: await supabase.auth.signInWithEmailAndPassword({email, password}) - signUp: await supabase.auth.signUpWithEmailAndPassword({email, password}) • After signup, insert KYC ID into profiles table if provided • Show error messages for invalid credentials • Redirect to "/" on success • Beautiful UI with Tailwind CSS (gradient card, smooth animations) • Client component (use "use client" directive) Include all error handling and loading states.
```

5.4 Marketplace/Home Page

PROMPT #4:

```
Create the main marketplace page at app/page.tsx Requirements: • Fetch all unsold coupons: const { data } = await supabase.from('coupons').select('*').eq('is_sold', false).order('created_at', { ascending: false }) • Display in a responsive grid (3 columns desktop, 1 mobile) • Each coupon card shows: - Title (bold, large) - Description (truncated to 2 lines) - Price in credits (with coin icon) - Expiry date (if available) - Category badge - "Buy Now" button • Search bar at top (filter by title) • Category filter dropdown • Loading skeleton while fetching • Empty state if no coupons found • Modern card design with hover effects • Use lucide-react icons This should be a server component that fetches data on the server.
```

5.5 Sell Coupon Page

PROMPT #5:

```
Create a "Sell Coupon" page at app/sell/page.tsx Requirements: • Form with fields: - Title (required, text input) - Description (textarea) - Coupon Code (required, text input with "show/hide" toggle) - Price in Credits (required, number input, min 1) - Category (select dropdown: Food, Fashion, Travel, Tech, Other) - Expiry Date (date picker, optional) - Image URL (optional, text input) • On submit: 1. Get current user ID: const { data: { user } } = await supabase.auth.getUser() 2. Insert into coupons table with seller_id = user.id 3. Show success toast/alert 4. Redirect to "/" (marketplace) • Form validation (all required fields) • Beautiful form UI with Tailwind CSS • Client component with useState for form data • Error handling Provide complete code with all imports.
```

5.6 Buy Transaction Logic (Server Action)

PROMPT #6 (CRITICAL):

```
Create a server action at actions/buyCoupon.ts for purchasing coupons. This is the MOST CRITICAL function. Requirements: export async function buyCoupon(couponId: string, buyerId: string) { 1. Fetch the coupon details (price, seller_id, is_sold) 2. Check if coupon is already sold (return error if yes) 3. Fetch buyer's credit balance from profiles 4. Check if buyer has enough credits (return error if not) 5. Use Supabase RPC or multiple updates to: a. Deduct credits from buyer b. Add credits to seller c. Update coupon: is_sold = true, buyer_id = buyerId d. Insert transaction record 6. Return { success: true, code: coupon.code } if successful 7. Return { success: false, error: "message" } if failed IMPORTANT: All database operations must succeed or fail together (use Supabase transactions if possible, or handle rollback manually) Include TypeScript types and comprehensive error handling. } Also provide a client component that: • Accepts couponId as prop • Calls the buyCoupon action onClick • Shows loading state during purchase • Displays modal/alert with coupon code on success • Shows error message on failure
```

5.7 User Profile/Dashboard Page

PROMPT #7:

```
Create a user profile page at app/profile/page.tsx Requirements: • Display user info: - Email - Current credit balance (large, prominent) - Account created date • Two tabs/sections: 1. "My Listings" - Coupons I've listed (show active and sold separately) 2. "My Purchases" - Coupons I've bought (show code, title, purchase date) • For listings, show: - Title, Price, Status (Active/Sold) - Edit button (just placeholder for now) - Delete button for unsold coupons • For purchases, show: - Coupon code (revealed) - Where to use it - Expiry date • "Buy More Credits" button (placeholder - show alert saying "Coming soon!") • Beautiful card-based layout • Server component for data fetching • Use Supabase to fetch user's coupons and purchases Include loading states and empty states.
```

5.8 Real-Time Chat Page

PROMPT #8:

```
Create a chat interface at app/chat/[userId]/page.tsx Requirements: • Accept userId as dynamic route parameter • Fetch chat history between current user and target user • Display messages in chat bubbles (sent messages right-aligned with blue background, received messages left-aligned with gray) • Input box at bottom with send button • Real-time updates using Supabase Realtime: useEffect(() => { const subscription = supabase .channel('messages') .on('postgres_changes', { event: 'INSERT', schema: 'public', table: 'messages' }, (payload) => { // Add new message to state if it's for this conversation } ) .subscribe() return () => { subscription.unsubscribe() } }, []) • Send message function: - Insert into messages table - Clear input field • Auto-scroll to bottom when new message arrives • Show timestamp for each message • Client component with useState and useEffect • Beautiful chat UI with Tailwind CSS Provide complete code with all imports and types.
```

6. Phase 3: Integration (VS Code + GitHub)

6.1 Test Your App Locally

1. In VS Code terminal, run:
`npm run dev`
2. Open `http://localhost:3000` in browser
3. Test each feature:
 - ✓ Sign up with new account
 - ✓ Check if you got 50 credits
 - ✓ List a test coupon
 - ✓ View marketplace
 - ✓ Buy a coupon (with another account or same)
 - ✓ Check if credits transferred correctly
 - ✓ Send a chat message
4. Fix any errors (check browser console and terminal for errors)

6.2 Initialize Git Repository

1. In VS Code terminal, run:
`git init`
2. Create `.gitignore` file (should already exist). Ensure it includes:
`node_modules/`
`.next/`
`.env.local`
3. Add all files:
`git add .`
4. Make first commit:
`git commit -m "Initial commit - CouponSwap MVP"`

6.3 Push to GitHub

1. Go to `https://github.com` and create new repository:
 - Name: `coupon-swap`
 - Description: P2P Coupon Exchange Marketplace
 - Keep it Public (for portfolio)
 - Don't initialize with README (we already have code)
2. Copy the repository URL (should look like):
`https://github.com/yourusername/coupon-swap.git`
3. In VS Code terminal, run:
`git branch -M main`
`git remote add origin https://github.com/yourusername/coupon-swap.git`
`git push -u origin main`
4. Refresh GitHub page - you should see your code!

7. Phase 4: Deployment (Vercel)

7.1 Connect Vercel to GitHub

1. Go to <https://vercel.com/dashboard>
2. Click 'Add New Project'
3. Click 'Import Git Repository'
4. Select your 'coupon-swap' repository
5. Vercel will auto-detect it's a Next.js app

7.2 Configure Environment Variables

CRITICAL STEP: Before clicking Deploy, you MUST add your environment variables. Click 'Environment Variables' section and add these:

Variable Name	Value
NEXT_PUBLIC_SUPABASE_URL	(Paste from Supabase dashboard)
NEXT_PUBLIC_SUPABASE_ANON_KEY	(Paste from Supabase dashboard)

Make sure to select "Production", "Preview", and "Development" for all variables.

7.3 Deploy!

1. Click 'Deploy' button
2. Wait 2-3 minutes while Vercel builds your app
3. You'll see a success screen with your live URL:
<https://coupon-swap-xxxxx.vercel.app>
4. Click the URL to see your live app!
5. Test all features on the live site
6. Share the link with friends and add to resume/portfolio

7.4 Continuous Deployment

Now, every time you push to GitHub, Vercel automatically rebuilds and deploys your app. This means you can keep improving your project and changes go live instantly!

8. Phase 5: AI Features (Optional - Portfolio Boost)

8.1 AI Coupon Scanner (OCR)

This feature lets users upload a screenshot of a coupon, and AI extracts the code automatically. Use Google Gemini Vision API for this.

PROMPT FOR GEMINI:

```
Add an image upload feature to my app/sell/page.tsx: 1. Add a file input for image upload (accept jpg, png, pdf) 2. When user selects image, send it to Google Gemini Vision API: - Use fetch() to call Gemini API - Prompt: "Extract the coupon code, brand name, discount amount, and expiry date from this image. Return as JSON." 3. Parse the API response and auto-fill the form fields 4. Show a loading spinner while processing 5. Handle errors (invalid image, no coupon detected) Provide complete code including: • File upload handling • API call to Gemini • Form auto-fill logic • Environment variable for GEMINI_API_KEY Also tell me how to get a free Gemini API key.
```

8.2 Recommendation Engine (Python ML)

Build a simple collaborative filtering recommendation system that suggests coupons based on user purchase history.

```
# recommendation_engine.py
# Install: pip install pandas scikit-learn supabase

import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity
from supabase import create_client

# Initialize Supabase
supabase = create_client(
    "YOUR_SUPABASE_URL",
    "YOUR_SUPABASE_ANON_KEY"
)

def get_recommendations(user_id):
    # Fetch all transactions
    transactions = supabase.table('transactions').select('*').execute()
    df = pd.DataFrame(transactions.data)

    # Create user-item matrix (users vs. coupon categories)
    user_category_matrix = pd.crosstab(
        df['buyer_id'],
        df['coupon_category']  # You'll need to join with coupons table
    )

    # Calculate similarity between users
    similarity = cosine_similarity(user_category_matrix)
    similarity_df = pd.DataFrame(
        similarity,
        index=user_category_matrix.index,
        columns=user_category_matrix.index
    )

    # Find similar users
    if user_id not in similarity_df.index:
        return ["No purchase history yet"]

    similar_users = similarity_df[user_id].sort_values(
        ascending=False
```

```
)[:4] # Top 3 similar users

# Get coupons they bought
recommendations = []
for similar_user in similar_users.index:
    user_coupons = df[df['buyer_id'] == similar_user]['coupon_id']
    recommendations.extend(user_coupons.tolist())

return list(set(recommendations))[:5] # Top 5 unique

# Test
print(get_recommendations("user-uuid-here"))
```

You can deploy this Python script as a separate API using Render.com or Railway.app (both have free tiers) and call it from your Next.js app to display recommendations on the homepage.

9. Troubleshooting Common Issues

Issue: Supabase returns "Row Level Security" error

- Make sure you ran the complete SQL script with all RLS policies
- Check if you are logged in (JWT token present)
- Verify the policy allows the operation you are trying

Issue: Environment variables not working on Vercel

- Did you add them in Vercel dashboard (NOT in code)?
- Did you redeploy after adding variables?
- Variable names must be EXACT (case-sensitive)

Issue: "Module not found" error

- Run: npm install
- Delete node_modules and package-lock.json, then npm install again
- Check for typos in import statements

Issue: Buy button not working / Credits not updating

- Check browser console for errors
- Verify the buyCoupon server action is called correctly
- Add console.log statements to debug
- Check Supabase logs for failed queries

Issue: Real-time chat not updating

- Verify Supabase Realtime is enabled in project settings
- Check the channel subscription code
- Make sure you are subscribed to correct table/filters

10. Quick Reference: All Prompts at a Glance

Copy this page to have all prompts in one place. Use them sequentially in Google Gemini or Claude AI.

Prompt 1	Initialize Supabase client (lib/supabase.ts) and types (types/index.ts)
Prompt 2	Create Navbar component (components/Navbar.tsx)
Prompt 3	Create Login/Signup page (app/login/page.tsx)
Prompt 4	Create Marketplace page (app/page.tsx)
Prompt 5	Create Sell Coupon page (app/sell/page.tsx)
Prompt 6	Create Buy transaction logic (actions/buyCoupon.ts + BuyButton)
Prompt 7	Create Profile page (app/profile/page.tsx)
Prompt 8	Create Chat interface (app/chat/[userId]/page.tsx)

Final Checklist Before Submission/Demo

- ✓ All core features working (auth, listing, buying, chat)
- ✓ Mobile responsive (test on phone)
- ✓ No console errors on production site
- ✓ README.md added to GitHub with project description and screenshots
- ✓ Environment variables configured on Vercel
- ✓ Test with 2-3 users to verify multi-user flow
- ✓ Recorded a 2-minute demo video (use OBS or Loom)
- ✓ Added project to portfolio/resume with live link

Congratulations!

You have now built a full-stack, production-ready web application with AI features. This project demonstrates your skills in React, Next.js, databases, authentication, real-time communication, and AI integration. Make sure to showcase it prominently in your resume and LinkedIn profile!