GrimForge Site - Comprehensive Handover Document

Project Overview & Current State

Project Information

- **Repository**: TGreen87/grimforge-site
- **Current Branch**: feat/next15-migration
- **Live Deployment**: https://deploy-preview-4–obsidianriterecords.netlify.app/
- **Owner Email**: arg@obsidianriterecords.com
- **Last Updated**: August 28, 2025

Current Tech Stack

- **Framework**: Next.js 15.5.0 (App Router)
- **React**: 18.3.1 (with React 19 RC support)
- **Database**: Supabase (PostgreSQL with pgvector extension)
- **Authentication**: Supabase Auth (ready for NextAuth v5 migration)
- **Admin Panel**: Refine.dev with Antd
- **Styling**: Tailwind CSS with shadcn/ui components
- **State Management**: TanStack Query v5
- **Payment Processing**: Stripe
- **Deployment**: Netlify
- **Testing**: Vitest + Playwright

Project Status Summary

✅ **Completed Features:**
- Next.js 15 migration with App Router
- Supabase database and authentication setup
- Admin panel with Refine.dev integration
- Basic e-commerce structure (products, categories)
- Stripe payment integration
- Comprehensive test suites (unit + e2e)
- Production deployment pipeline

🔄 **Current State:**
- Site is fully functional and deployed
- Admin authentication working
- Database schema established
- Performance optimizations partially implemented

4 High-Impact Features Roadmap

Priority Order & Dependencies

1. **Google OAuth Implementation** (1-2 weeks) - Foundation for personalization

2. **Performance Optimization** (1-2 weeks) - Can run parallel with OAuth
3. **Enhanced Product Discovery** (2-3 weeks) - Depends on performance optimizations
4. **Personalized Shopping** (2-3 weeks) - Depends on OAuth and product discovery

---

Feature 1: Google OAuth Implementation

Overview

Migrate from Supabase Auth to NextAuth v5 (Auth.js) with Google OAuth provider for enhanced user experience and better integration with personalization features.

Technical Specifications

Dependencies to Install

```
npm install next-auth@beta @auth/supabase-adapter
```

Environment Variables Required

```
# Add to .env.local
AUTH_SECRET=your-secret-key-here
AUTH_GOOGLE_ID=your-google-client-id
AUTH_GOOGLE_SECRET=your-google-client-secret
AUTH_TRUST_HOST=true
```

File Structure Changes

```
/auth.config.ts          # Edge-compatible config
/auth.ts                 # Main Auth.js configuration
/middleware.ts           # Route protection
/app/api/auth/[...nextauth]/route.ts  # API routes
/app/login/page.tsx      # Custom login page
/lib/auth-helpers.ts     # Auth utility functions
```

Implementation Steps

Step 1: Create Auth Configuration Files

Create `/auth.config.ts`:

```
import type { NextAuthConfig } from "next-auth";
import Google from "next-auth/providers/google";

export const authConfig = {
  providers: [
    Google({
      clientId: process.env.AUTH_GOOGLE_ID,
      clientSecret: process.env.AUTH_GOOGLE_SECRET,
      authorization: {
        params: {
          prompt: "consent",
          access_type: "offline",
          response_type: "code",
        },
      },
    }),
  ],
  pages: {
    signIn: "/login",
    error: "/auth/error"
  },
  callbacks: {
    authorized({ auth, request: { nextUrl } }) {
      const isLoggedIn = !!auth?.user;
      const isOnDashboard = nextUrl.pathname.startsWith('/dashboard');
      const isOnAdmin = nextUrl.pathname.startsWith('/admin');

      if (isOnDashboard || isOnAdmin) {
        if (isLoggedIn) return true;
        return false; // Redirect unauthenticated users to login page
      }
      return true;
    },
  },
} satisfies NextAuthConfig;
```

Create `/auth.ts`:

```
import NextAuth from "next-auth";
import { SupabaseAdapter } from "@auth/supabase-adapter";
import { createClient } from "@supabase/supabase-js";
import { authConfig } from "./auth.config";

const supabase = createClient(
  process.env.NEXT_PUBLIC_SUPABASE_URL!,
  process.env.SUPABASE_SERVICE_ROLE_KEY!
);

export const { auth, handlers, signIn, signOut } = NextAuth({
  ...authConfig,
  adapter: SupabaseAdapter(supabase),
  session: { strategy: "jwt" },
  callbacks: {
    async jwt({ token, account, profile }) {
      if (account?.provider === "google") {
        token.email = profile?.email;
        token.picture = profile?.picture;
        token.name = profile?.name;
      }
      return token;
    },
    async session({ session, token }) {
      if (token) {
        session.user.id = token.sub!;
        session.user.email = token.email!;
        session.user.name = token.name!;
        session.user.image = token.picture as string;
      }
      return session;
    },
  },
});
```

Step 2: Set Up API Routes

Create `/app/api/auth/[...nextauth]/route.ts`:

```
import { handlers } from "@/auth";
export const { GET, POST } = handlers;
```

Step 3: Create Middleware for Route Protection

Create `/middleware.ts`:

```
import { auth } from "@/auth";
import { NextResponse } from "next/server";

export default auth((req) => {
  const isLoggedIn = !!req.auth;
  const { pathname } = req.nextUrl;

  // Protect admin routes
  if (pathname.startsWith('/admin') && !isLoggedIn) {
    return NextResponse.redirect(new URL('/login', req.url));
  }

  // Protect dashboard routes
  if (pathname.startsWith('/dashboard') && !isLoggedIn) {
    return NextResponse.redirect(new URL('/login', req.url));
  }

  return NextResponse.next();
});

export const config = {
  matcher: ['/admin/:path*', '/dashboard/:path*']
};
```

Step 4: Create Login Page

Create `/app/login/page.tsx`:

```
"use client";
import { signIn, useSession } from "next-auth/react";
import { useRouter } from "next/navigation";
import { useEffect } from "react";
import { Button } from "@/components/ui/button";
import { Card, CardContent, CardDescription, CardHeader, CardTitle } from "@/compon-
ents/ui/card";

export default function LoginPage() {
  const { data: session, status } = useSession();
  const router = useRouter();

  useEffect(() => {
    if (session) {
      router.push('/dashboard');
    }
  }, [session, router]);

  if (status === "loading") {
    return <div className="flex justify-center items-center min-h-screen">Loading...</
div>;
  }

  return (
    <div className="flex justify-center items-center min-h-screen bg-gray-50">
      <Card className="w-full max-w-md">
        <CardHeader>
          <CardTitle>Welcome to GrimForge</CardTitle>
          <CardDescription>Sign in to access your account</CardDescription>
        </CardHeader>
        <CardContent>
          <Button
            onClick={() => signIn("google", { callbackUrl: "/dashboard" })}
            className="w-full"
            size="lg"
          >
            <svg className="w-5 h-5 mr-2" viewBox="0 0 24 24">
              <path fill="currentColor" d="M22.56 12.25c0-.78-.07-1.53-.
2-2.25H12v4.26h5.92c-.26 1.37-1.04 2.53-2.21 3.31v2.77h3.57c2.08-1.92 3.28-4.74
3.28-8.09z"/>
              <path fill="currentColor" d="M12 23c2.97 0 5.46-.98
7.28-2.66l-3.57-2.77c-.98.66-2.23 1.06-3.71 1.06-2.86
0-5.29-1.93-6.16-4.53H2.18v2.84C3.99 20.53 7.7 23 12 23z"/>
              <path fill="currentColor" d="M5.84 14.09c-.22-.66-.35-1.36-.35-2.09s.
13-1.43.35-2.09V7.07H2.18C1.43 8.55 1 10.22 1 12s.43 3.45 1.18 4.93l2.85-2.22.81-.62z"/
>
              <path fill="currentColor" d="M12 5.38c1.62 0 3.06.56 4.21
1.64l3.15-3.15C17.45 2.09 14.97 1 12 1 7.7 1 3.99 3.47 2.18 7.07l3.66 2.84c.87-2.6
3.3-4.53 6.16-4.53z"/>
            </svg>
            Continue with Google
          </Button>
        </CardContent>
      </Card>
    </div>
  );
}
```

Step 5: Update Root Layout

Update `/app/layout.tsx`:

```tsx
import { SessionProvider } from "next-auth/react";
import { auth } from "@/auth";

export default async function RootLayout({
  children,
}: {
  children: React.ReactNode;
}) {
  const session = await auth();

  return (
    <html lang="en">
      <body>
        <SessionProvider session={session}>
          {children}
        </SessionProvider>
      </body>
    </html>
  );
}
```

Google Cloud Console Setup

1. Go to Google Cloud Console (https://console.cloud.google.com/)
2. Create a new project or select existing one
3. Enable Google+ API
4. Go to “Credentials” → “Create Credentials” → “OAuth 2.0 Client IDs”
5. Set application type to “Web application”
6. Add authorized redirect URIs:
   - `http://localhost:3000/api/auth/callback/google` (development)
   - `https://your-domain.com/api/auth/callback/google` (production)

Testing Procedures

Manual Testing Checklist

- [ ] Google OAuth login flow works
- [ ] User session persists across page refreshes
- [ ] Protected routes redirect to login when unauthenticated
- [ ] Logout functionality works correctly
- [ ] User data is stored in Supabase
- [ ] Admin panel access is properly protected

Automated Tests

```
// tests/auth.test.ts
import { test, expect } from '@playwright/test';

test.describe('Authentication', () => {
  test('should redirect to login when accessing protected route', async ({ page }) => {
    await page.goto('/admin');
    await expect(page).toHaveURL(/.*login/);
  });

  test('should allow access after login', async ({ page }) => {
    // Mock Google OAuth for testing
    await page.goto('/login');
    // Add test implementation
  });
});
```

Success Criteria

- [ ] Users can sign in with Google OAuth
- [ ] Session management works correctly
- [ ] Protected routes are secured
- [ ] User data syncs with Supabase
- [ ] Admin panel integration maintained
- [ ] No breaking changes to existing functionality

Feature 2: Performance Optimization with Next.js 15

Overview

Leverage Next.js 15's new performance features including Partial Prerendering (PPR), Turbopack, and optimized caching strategies.

Technical Specifications

Next.js 15 Features to Implement

1. **Partial Prerendering (PPR)** - Static shell with dynamic content
2. **Turbopack** - Faster development and build times
3. **Optimized Caching** - Smart cache invalidation
4. **React 19 Integration** - Automatic optimizations
5. **Static Route Indicator** - Development insights

Implementation Steps

Step 1: Enable Experimental Features

Update `/next.config.mjs`:

```
/** @type {import('next').NextConfig} */
const nextConfig = {
  reactStrictMode: true,
  experimental: {
    // Enable Partial Prerendering
    ppr: 'incremental',
    // Enable after API for background tasks
    after: true,
    // Enable Turbopack for development
    turbo: {
      rules: {
        '*.svg': {
          loaders: ['@svgr/webpack'],
          as: '*.js',
        },
      },
    },
    // Optimize CSS
    optimizeCss: true,
    // Enable static generation optimizations
    staticGenerationAsyncStorage: true,
  },
  // Enable Turbopack for production builds
  turbopack: true,
  images: {
    domains: ['shbalyvvquvtvnkrsxtx.supabase.co'],
    formats: ['image/webp', 'image/avif'],
  },
  // Optimize bundle
  compiler: {
    removeConsole: process.env.NODE_ENV === 'production',
  },
  // Enable React 19 features
  typescript: {
    ignoreBuildErrors: false, // Re-enable after fixing TS issues
  },
  eslint: {
    ignoreDuringBuilds: false, // Re-enable after fixing ESLint issues
  },
};

export default nextConfig;
```

Step 2: Implement Partial Prerendering

Create `/app/products/[id]/page.tsx` with PPR:

```tsx
import { Suspense } from 'react';
import { unstable_noStore as noStore } from 'next/cache';
import ProductDetails from '@/components/ProductDetails';
import ProductRecommendations from '@/components/ProductRecommendations';
import ProductSkeleton from '@/components/ProductSkeleton';

// Static part - prerendered
export default async function ProductPage({ params }: { params: { id: string } }) {
  return (
    <div className="container mx-auto px-4 py-8">
      {/* Static shell */}
      <div className="grid grid-cols-1 lg:grid-cols-2 gap-8">
        {/* Dynamic product details */}
        <Suspense fallback={<ProductSkeleton />}>
          <ProductDetailsWrapper productId={params.id} />
        </Suspense>

        {/* Static sidebar */}
        <div className="space-y-6">
          <div className="bg-gray-50 p-6 rounded-lg">
            <h3 className="text-lg font-semibold mb-4">Product Information</h3>
            {/* Static content */}
          </div>

          {/* Dynamic recommendations */}
          <Suspense fallback={<div>Loading recommendations...</div>}>
            <ProductRecommendationsWrapper productId={params.id} />
          </Suspense>
        </div>
      </div>
    </div>
  );
}

// Dynamic component that opts out of static generation
async function ProductDetailsWrapper({ productId }: { productId: string }) {
  noStore(); // Opt out of static generation for this component

  const product = await fetchProduct(productId);
  return <ProductDetails product={product} />;
}

async function ProductRecommendationsWrapper({ productId }: { productId: string }) {
  noStore(); // Dynamic recommendations based on user behavior

  const recommendations = await fetchRecommendations(productId);
  return <ProductRecommendations products={recommendations} />;
}
```

Step 3: Optimize Caching Strategy

Create `/lib/cache.ts`:

```
import { unstable_cache } from 'next/cache';
import { unstable_after as after } from 'next/server';

// Cache product data with smart invalidation
export const getCachedProduct = unstable_cache(
  async (productId: string) => {
    const product = await fetchProductFromDB(productId);

    // Background task to update analytics
    after(() => {
      updateProductViewCount(productId);
    });

    return product;
  },
  ['product'],
  {
    tags: ['products'],
    revalidate: 3600, // 1 hour
  }
);

// Cache category data
export const getCachedCategories = unstable_cache(
  async () => {
    return await fetchCategoriesFromDB();
  },
  ['categories'],
  {
    tags: ['categories'],
    revalidate: 86400, // 24 hours
  }
);

// Invalidate cache when products are updated
export async function revalidateProductCache(productId?: string) {
  if (productId) {
    revalidateTag(`product-${productId}`);
  } else {
    revalidateTag('products');
  }
}
```

Step 4: Implement Background Tasks with `after`

Create `/app/api/products/[id]/view/route.ts`:

```
import { unstable_after as after } from 'next/server';
import { NextRequest, NextResponse } from 'next/server';

export async function POST(
  request: NextRequest,
  { params }: { params: { id: string } }
) {
  const productId = params.id;

  // Immediate response
  const response = NextResponse.json({ success: true });

  // Background tasks that don't block the response
  after(async () => {
    // Update view count
    await updateProductViewCount(productId);

    // Log analytics
    await logProductView(productId, request);

    // Update recommendations
    await updateRecommendationCache(productId);
  });

  return response;
}
```

Step 5: Optimize Images and Assets

Create `/components/OptimizedImage.tsx`:

```tsx
import Image from 'next/image';
import { useState } from 'react';

interface OptimizedImageProps {
  src: string;
  alt: string;
  width: number;
  height: number;
  priority?: boolean;
  className?: string;
}

export default function OptimizedImage({
  src,
  alt,
  width,
  height,
  priority = false,
  className = '',
}: OptimizedImageProps) {
  const [isLoading, setIsLoading] = useState(true);

  return (
    <div className={`relative overflow-hidden ${className}`}>
      <Image
        src={src}
        alt={alt}
        width={width}
        height={height}
        priority={priority}
        className={`transition-opacity duration-300 ${
          isLoading ? 'opacity-0' : 'opacity-100'
        }`}
        onLoad={() => setIsLoading(false)}
        sizes="(max-width: 768px) 100vw, (max-width: 1200px) 50vw, 33vw"
        placeholder="blur"
        blurDataURL="-
BAYGBQYHBwYIChAKCgkJChQODwwQFxQYGBcUFhYaHSUfGhsjHBYWICwgIyYnKSopGR8tMC0oMCUoKSj/
2wBDAQcHBwoIChMKChMoGhYaKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoK-
CgoKCj/wAARCAABAAEDASIAAhEBAxEB/8QAFQABAQAAAAAAAAAAAAAAAAAAAAv/xAAUEAE-
AAAAAAAAAAAAAAAAAAAA/8QAFQEBAQAAAAAAAAAAAAAAAAAAAX/xAAUEQEAAAAAAAAAAAAAAAAAAAA/
9oADAMBAAIRAxEAPwCdABmX/9k="
      />
      {isLoading && (
        <div className="absolute inset-0 bg-gray-200 animate-pulse" />
      )}
    </div>
  );
}
```

Performance Monitoring

Create `/lib/performance.ts`:

```
import { unstable_after as after } from 'next/server';

export function trackPerformance(metricName: string, value: number) {
  after(() => {
    // Send to analytics service
    fetch('/api/analytics/performance', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({
        metric: metricName,
        value,
        timestamp: Date.now(),
      }),
    });
  });
}

export function measurePageLoad() {
  if (typeof window !== 'undefined') {
    window.addEventListener('load', () => {
      const navigation = performance.getEntriesByType('navigation')[0] as Performan-
ceNavigationTiming;
      const loadTime = navigation.loadEventEnd - navigation.loadEventStart;
      trackPerformance('page_load_time', loadTime);
    });
  }
}
```

Testing Performance

Create `/tests/performance.test.ts`:

```
import { test, expect } from '@playwright/test';

test.describe('Performance', () => {
  test('should load homepage within 2 seconds', async ({ page }) => {
    const startTime = Date.now();
    await page.goto('/');
    await page.waitForLoadState('networkidle');
    const loadTime = Date.now() - startTime;

    expect(loadTime).toBeLessThan(2000);
  });

  test('should have good Core Web Vitals', async ({ page }) => {
    await page.goto('/');

    const metrics = await page.evaluate(() => {
      return new Promise((resolve) => {
        new PerformanceObserver((list) => {
          const entries = list.getEntries();
          resolve(entries.map(entry => ({
            name: entry.name,
            value: entry.value,
          })));
        }).observe({ entryTypes: ['measure', 'navigation'] });
      });
    });

    // Add assertions for LCP, FID, CLS
  });
});
```

Success Criteria

- [ ] Page load times improved by 40%+
- [ ] Build times reduced with Turbopack
- [ ] Core Web Vitals scores in “Good” range
- [ ] PPR working for product pages
- [ ] Background tasks not blocking responses
- [ ] Static route indicator showing optimizations

Feature 3: Enhanced Product Discovery

Overview

Implement semantic search using Supabase pgvector, advanced filtering, and AI-powered recommendations for superior product discovery.

Technical Specifications

Database Schema Updates

Enable pgvector extension:

```
-- Run in Supabase SQL Editor
CREATE EXTENSION IF NOT EXISTS vector WITH SCHEMA extensions;
```

Update products table:

```sql
-- Add vector column for embeddings
ALTER TABLE products
ADD COLUMN embedding vector(384),
ADD COLUMN search_vector tsvector,
ADD COLUMN tags text[],
ADD COLUMN attributes jsonb DEFAULT '{}';

-- Create indexes for performance
CREATE INDEX ON products USING ivfflat (embedding vector_cosine_ops) WITH (lists = 100)
;
CREATE INDEX ON products USING gin(search_vector);
CREATE INDEX ON products USING gin(tags);
CREATE INDEX ON products USING gin(attributes);

-- Create function for semantic search
CREATE OR REPLACE FUNCTION match_products (
  query_embedding vector(384),
  match_threshold float DEFAULT 0.78,
  match_count int DEFAULT 10,
  filter_category text DEFAULT NULL,
  min_price numeric DEFAULT NULL,
  max_price numeric DEFAULT NULL
) RETURNS TABLE (
  id bigint,
  title text,
  description text,
  price numeric,
  category text,
  image_url text,
  similarity float
) LANGUAGE sql AS $$
  SELECT
    p.id,
    p.title,
    p.description,
    p.price,
    p.category,
    p.image_url,
    1 - (p.embedding <=> query_embedding) as similarity
  FROM products p
  WHERE
    (filter_category IS NULL OR p.category = filter_category)
    AND (min_price IS NULL OR p.price >= min_price)
    AND (max_price IS NULL OR p.price <= max_price)
    AND p.embedding <=> query_embedding < (1 - match_threshold)
  ORDER BY p.embedding <=> query_embedding ASC
  LIMIT LEAST(match_count, 50);
$$;

-- Create function for hybrid search (semantic + keyword)
CREATE OR REPLACE FUNCTION hybrid_search_products (
  query_text text,
  query_embedding vector(384),
  match_threshold float DEFAULT 0.78,
  match_count int DEFAULT 10,
  semantic_weight float DEFAULT 0.7,
  keyword_weight float DEFAULT 0.3
) RETURNS TABLE (
  id bigint,
  title text,
  description text,
  price numeric,
```

```
  category text,
  image_url text,
  combined_score float
) LANGUAGE sql AS $$
  WITH semantic_search AS (
    SELECT
      p.id,
      p.title,
      p.description,
      p.price,
      p.category,
      p.image_url,
      (1 - (p.embedding <=> query_embedding)) * semantic_weight as semantic_score
    FROM products p
    WHERE p.embedding <=> query_embedding < (1 - match_threshold)
  ),
  keyword_search AS (
    SELECT
      p.id,
      p.title,
      p.description,
      p.price,
      p.category,
      p.image_url,
      ts_rank(p.search_vector, plainto_tsquery(query_text)) * keyword_weight as keyword
_score
    FROM products p
    WHERE p.search_vector @@ plainto_tsquery(query_text)
  )
  SELECT
    COALESCE(s.id, k.id) as id,
    COALESCE(s.title, k.title) as title,
    COALESCE(s.description, k.description) as description,
    COALESCE(s.price, k.price) as price,
    COALESCE(s.category, k.category) as category,
    COALESCE(s.image_url, k.image_url) as image_url,
    COALESCE(s.semantic_score, 0) + COALESCE(k.keyword_score, 0) as combined_score
  FROM semantic_search s
  FULL OUTER JOIN keyword_search k ON s.id = k.id
  ORDER BY combined_score DESC
  LIMIT match_count;
$$;
```

Implementation Steps

Step 1: Set Up Embedding Generation

Create `/lib/embeddings.ts`:

```
import { pipeline } from '@xenova/transformers';

// Initialize the embedding model
let embedder: any = null;

async function getEmbedder() {
  if (!embedder) {
    embedder = await pipeline('feature-extraction', 'Supabase/gte-small');
  }
  return embedder;
}

export async function generateEmbedding(text: string): Promise<number[]> {
  try {
    const model = await getEmbedder();
    const output = await model(text, { pooling: 'mean', normalize: true });
    return Array.from(output.data);
  } catch (error) {
    console.error('Error generating embedding:', error);
    throw new Error('Failed to generate embedding');
  }
}

export async function generateProductEmbedding(product: {
  title: string;
  description: string;
  category: string;
  tags?: string[];
}): Promise<number[]> {
  const text = [
    product.title,
    product.description,
    product.category,
    ...(product.tags || [])
  ].join(' ');

  return generateEmbedding(text);
}
```

Step 2: Create Search API

Create `/app/api/search/route.ts`:

```
import { NextRequest, NextResponse } from 'next/server';
import { createClient } from '@/lib/supabase/server';
import { generateEmbedding } from '@/lib/embeddings';

export async function GET(request: NextRequest) {
  const { searchParams } = new URL(request.url);
  const query = searchParams.get('q');
  const category = searchParams.get('category');
  const minPrice = searchParams.get('min_price');
  const maxPrice = searchParams.get('max_price');
  const searchType = searchParams.get('type') || 'hybrid'; // semantic, keyword, hybrid
  const limit = parseInt(searchParams.get('limit') || '10');

  if (!query) {
    return NextResponse.json({ error: 'Query parameter is required' }, { status: 400 })
;
  }

  try {
    const supabase = createClient();
    let results;

    if (searchType === 'semantic' || searchType === 'hybrid') {
      const embedding = await generateEmbedding(query);

      if (searchType === 'semantic') {
        const { data, error } = await supabase.rpc('match_products', {
          query_embedding: embedding,
          match_threshold: 0.78,
          match_count: limit,
          filter_category: category,
          min_price: minPrice ? parseFloat(minPrice) : null,
          max_price: maxPrice ? parseFloat(maxPrice) : null,
        });

        if (error) throw error;
        results = data;
      } else {
        // Hybrid search
        const { data, error } = await supabase.rpc('hybrid_search_products', {
          query_text: query,
          query_embedding: embedding,
          match_threshold: 0.78,
          match_count: limit,
          semantic_weight: 0.7,
          keyword_weight: 0.3,
        });

        if (error) throw error;
        results = data;
      }
    } else {
      // Keyword search only
      let queryBuilder = supabase
        .from('products')
        .select('*')
        .textSearch('search_vector', query)
        .limit(limit);

      if (category) queryBuilder = queryBuilder.eq('category', category);
      if (minPrice) queryBuilder = queryBuilder.gte('price', parseFloat(minPrice));
      if (maxPrice) queryBuilder = queryBuilder.lte('price', parseFloat(maxPrice));
```

```
      const { data, error } = await queryBuilder;
      if (error) throw error;
      results = data;
    }

    return NextResponse.json({
      results,
      query,
      searchType,
      count: results?.length || 0,
    });
  } catch (error) {
    console.error('Search error:', error);
    return NextResponse.json(
      { error: 'Search failed' },
      { status: 500 }
    );
  }
}
```

Step 3: Create Advanced Search Component

Create `/components/search/AdvancedSearch.tsx`:

```
'use client';
import { useState, useEffect, useMemo } from 'react';
import { useDebounce } from '@/hooks/useDebounce';
import { Input } from '@/components/ui/input';
import { Button } from '@/components/ui/button';
import { Select, SelectContent, SelectItem, SelectTrigger, SelectValue } from '@/com-
ponents/ui/select';
import { Slider } from '@/components/ui/slider';
import { Badge } from '@/components/ui/badge';
import { Search, Filter, X } from 'lucide-react';

interface SearchFilters {
  category?: string;
  minPrice?: number;
  maxPrice?: number;
  tags?: string[];
  searchType: 'semantic' | 'keyword' | 'hybrid';
}

interface Product {
  id: number;
  title: string;
  description: string;
  price: number;
  category: string;
  image_url: string;
  similarity?: number;
  combined_score?: number;
}

export default function AdvancedSearch() {
  const [query, setQuery] = useState('');
  const [filters, setFilters] = useState<SearchFilters>({
    searchType: 'hybrid',
  });
  const [results, setResults] = useState<Product[]>([]);
  const [loading, setLoading] = useState(false);
  const [showFilters, setShowFilters] = useState(false);

  const debouncedQuery = useDebounce(query, 300);

  const searchParams = useMemo(() => {
    const params = new URLSearchParams();
    if (debouncedQuery) params.set('q', debouncedQuery);
    if (filters.category) params.set('category', filters.category);
    if (filters.minPrice) params.set('min_price', filters.minPrice.toString());
    if (filters.maxPrice) params.set('max_price', filters.maxPrice.toString());
    params.set('type', filters.searchType);
    return params;
  }, [debouncedQuery, filters]);

  useEffect(() => {
    if (!debouncedQuery) {
      setResults([]);
      return;
    }

    const performSearch = async () => {
      setLoading(true);
      try {
        const response = await fetch(`/api/search?${searchParams}`);
        const data = await response.json();
```

```
        setResults(data.results || []);
      } catch (error) {
        console.error('Search failed:', error);
        setResults([]);
      } finally {
        setLoading(false);
      }
    };

    performSearch();
  }, [searchParams, debouncedQuery]);

  const clearFilters = () => {
    setFilters({ searchType: 'hybrid' });
  };

  return (
    <div className="w-full max-w-4xl mx-auto space-y-6">
      {/* Search Input */}
      <div className="relative">
        <Search className="absolute left-3 top-1/2 transform -translate-y-1/2 text-
gray-400 w-5 h-5" />
        <Input
          type="text"
          placeholder="Search for products..."
          value={query}
          onChange={(e) => setQuery(e.target.value)}
          className="pl-10 pr-12 h-12 text-lg"
        />
        <Button
          variant="ghost"
          size="sm"
          onClick={() => setShowFilters(!showFilters)}
          className="absolute right-2 top-1/2 transform -translate-y-1/2"
        >
          <Filter className="w-4 h-4" />
        </Button>
      </div>

      {/* Filters */}
      {showFilters && (
        <div className="bg-gray-50 p-4 rounded-lg space-y-4">
          <div className="flex items-center justify-between">
            <h3 className="font-semibold">Filters</h3>
            <Button variant="ghost" size="sm" onClick={clearFilters}>
              <X className="w-4 h-4 mr-1" />
              Clear
            </Button>
          </div>

          <div className="grid grid-cols-1 md:grid-cols-3 gap-4">
            {/* Search Type */}
            <div>
              <label className="block text-sm font-medium mb-2">Search Type</label>
              <Select
                value={filters.searchType}
                onValueChange={(value: 'semantic' | 'keyword' | 'hybrid') =>
                  setFilters({ ...filters, searchType: value })
                }
              >
                <SelectTrigger>
                  <SelectValue />
                </SelectTrigger>
```

```
                <SelectContent>
                  <SelectItem value="hybrid">Hybrid (Recommended)</SelectItem>
                  <SelectItem value="semantic">Semantic</SelectItem>
                  <SelectItem value="keyword">Keyword</SelectItem>
                </SelectContent>
              </Select>
            </div>

            {/* Category */}
            <div>
              <label className="block text-sm font-medium mb-2">Category</label>
              <Select
                value={filters.category || ''}
                onValueChange={(value) =>
                  setFilters({ ...filters, category: value || undefined })
                }
              >
                <SelectTrigger>
                  <SelectValue placeholder="All categories" />
                </SelectTrigger>
                <SelectContent>
                  <SelectItem value="">All categories</SelectItem>
                  <SelectItem value="electronics">Electronics</SelectItem>
                  <SelectItem value="clothing">Clothing</SelectItem>
                  <SelectItem value="books">Books</SelectItem>
                  <SelectItem value="home">Home & Garden</SelectItem>
                </SelectContent>
              </Select>
            </div>

            {/* Price Range */}
            <div>
              <label className="block text-sm font-medium mb-2">
                Price Range: ${filters.minPrice || 0} - ${filters.maxPrice || 1000}
              </label>
              <Slider
                value={[filters.minPrice || 0, filters.maxPrice || 1000]}
                onValueChange={([min, max]) =>
                  setFilters({ ...filters, minPrice: min, maxPrice: max })
                }
                max={1000}
                step={10}
                className="w-full"
              />
            </div>
          </div>
        )}

        {/* Search Results */}
        <div className="space-y-4">
          {loading && (
            <div className="text-center py-8">
              <div className="animate-spin rounded-full h-8 w-8 border-b-2 border-
gray-900 mx-auto"></div>
              <p className="mt-2 text-gray-600">Searching...</p>
            </div>
          )}

          {!loading && results.length === 0 && debouncedQuery && (
            <div className="text-center py-8 text-gray-600">
              No products found for "{debouncedQuery}"
            </div>
```

```
        )}

        {!loading && results.length > 0 && (
          <>
            <div className="flex items-center justify-between">
              <p className="text-sm text-gray-600">
                Found {results.length} products for "{debouncedQuery}"
              </p>
              <Badge variant="secondary">{filters.searchType} search</Badge>
            </div>

            <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-6">
              {results.map((product) => (
                <ProductCard key={product.id} product={product} />
              ))}
            </div>
          </>
        )}
      </div>
    </div>
  );
}

function ProductCard({ product }: { product: Product }) {
  return (
    <div className="bg-white rounded-lg shadow-md overflow-hidden hover:shadow-lg
transition-shadow">
      <img
        src={product.image_url}
        alt={product.title}
        className="w-full h-48 object-cover"
      />
      <div className="p-4">
        <h3 className="font-semibold text-lg mb-2 line-clamp-2">{product.title}</h3>
        <p className="text-gray-600 text-sm mb-3 line-clamp-3">{product.description}</
p>

        <div className="flex items-center justify-between">
          <span className="text-xl font-bold text-green-600">${product.price}</span>
          {(product.similarity || product.combined_score) && (
            <Badge variant="outline" className="text-xs">
              {Math.round((product.similarity || product.combined_score || 0) * 100)}%
match
            </Badge>
          )}
        </div>
      </div>
    </div>
  );
}
```

Step 4: Create Recommendation System

Create `/lib/recommendations.ts`:

```typescript
import { createClient } from '@/lib/supabase/server';

export interface RecommendationOptions {
  userId?: string;
  productId?: string;
  category?: string;
  limit?: number;
  type: 'similar' | 'collaborative' | 'trending' | 'personalized';
}

export async function getRecommendations(options: RecommendationOptions) {
  const supabase = createClient();
  const { userId, productId, category, limit = 10, type } = options;

  switch (type) {
    case 'similar':
      return getSimilarProducts(productId!, limit);

    case 'collaborative':
      return getCollaborativeRecommendations(userId!, limit);

    case 'trending':
      return getTrendingProducts(category, limit);

    case 'personalized':
      return getPersonalizedRecommendations(userId!, limit);

    default:
      throw new Error('Invalid recommendation type');
  }
}

async function getSimilarProducts(productId: string, limit: number) {
  const supabase = createClient();

  // Get the product's embedding
  const { data: product, error: productError } = await supabase
    .from('products')
    .select('embedding, category')
    .eq('id', productId)
    .single();

  if (productError || !product?.embedding) {
    throw new Error('Product not found or missing embedding');
  }

  // Find similar products
  const { data, error } = await supabase.rpc('match_products', {
    query_embedding: product.embedding,
    match_threshold: 0.7,
    match_count: limit + 1, // +1 to exclude the original product
    filter_category: product.category,
  });

  if (error) throw error;

  // Filter out the original product
  return data?.filter(p => p.id !== parseInt(productId)) || [];
}

async function getCollaborativeRecommendations(userId: string, limit: number) {
  const supabase = createClient();
```

```
  // This is a simplified collaborative filtering
  // In production, you'd use more sophisticated algorithms
  const { data, error } = await supabase
    .from('user_interactions')
    .select(`
      product_id,
      products (
        id,
        title,
        description,
        price,
        category,
        image_url
      )
    `)
    .eq('user_id', userId)
    .eq('interaction_type', 'purchase')
    .order('created_at', { ascending: false })
    .limit(limit);

  if (error) throw error;
  return data?.map(item => item.products) || [];
}

async function getTrendingProducts(category?: string, limit: number = 10) {
  const supabase = createClient();

  let query = supabase
    .from('products')
    .select('*')
    .order('view_count', { ascending: false })
    .limit(limit);

  if (category) {
    query = query.eq('category', category);
  }

  const { data, error } = await query;
  if (error) throw error;
  return data || [];
}

async function getPersonalizedRecommendations(userId: string, limit: number) {
  const supabase = createClient();

  // Get user's interaction history
  const { data: interactions, error: interactionError } = await supabase
    .from('user_interactions')
    .select('product_id, interaction_type, products(category, tags)')
    .eq('user_id', userId)
    .order('created_at', { ascending: false })
    .limit(50);

  if (interactionError) throw interactionError;

  // Analyze user preferences
  const categoryPreferences = new Map<string, number>();
  const tagPreferences = new Map<string, number>();

  interactions?.forEach(interaction => {
    const weight = interaction.interaction_type === 'purchase' ? 3 :
                   interaction.interaction_type === 'add_to_cart' ? 2 : 1;
```

```
    // Category preferences
    const category = interaction.products?.category;
    if (category) {
      categoryPreferences.set(category, (categoryPreferences.get(category) || 0) + weig
ht);
    }

    // Tag preferences
    const tags = interaction.products?.tags || [];
    tags.forEach(tag => {
      tagPreferences.set(tag, (tagPreferences.get(tag) || 0) + weight);
    });
  });

  // Get top categories and tags
  const topCategories = Array.from(categoryPreferences.entries())
    .sort(([,a], [,b]) => b - a)
    .slice(0, 3)
    .map(([category]) => category);

  const topTags = Array.from(tagPreferences.entries())
    .sort(([,a], [,b]) => b - a)
    .slice(0, 5)
    .map(([tag]) => tag);

  // Find products matching preferences
  let query = supabase
    .from('products')
    .select('*')
    .in('category', topCategories)
    .overlaps('tags', topTags)
    .limit(limit);

  const { data, error } = await query;
  if (error) throw error;
  return data || [];
}
```

Success Criteria

- [ ] Semantic search returns relevant results
- [ ] Advanced filtering works correctly
- [ ] Hybrid search combines semantic and keyword effectively
- [ ] Recommendations are personalized and accurate
- [ ] Search performance is under 500ms
- [ ] Vector embeddings are generated for all products

Feature 4: Personalized Shopping Experience

Overview

Create a comprehensive personalized shopping experience with browsing history, wishlist functionality, and AI-driven recommendations.

Technical Specifications

Database Schema for Personalization

```sql
-- User interactions table
CREATE TABLE user_interactions (
  id bigserial PRIMARY KEY,
  user_id uuid REFERENCES auth.users(id) ON DELETE CASCADE,
  product_id bigint REFERENCES products(id) ON DELETE CASCADE,
  interaction_type text NOT NULL CHECK (interaction_type IN ('view', 'add_to_cart', 'pu
rchase', 'wishlist_add', 'wishlist_remove')),
  session_id text,
  metadata jsonb DEFAULT '{}',
  created_at timestamp with time zone DEFAULT now()
);

-- Wishlist table
CREATE TABLE wishlists (
  id bigserial PRIMARY KEY,
  user_id uuid REFERENCES auth.users(id) ON DELETE CASCADE,
  product_id bigint REFERENCES products(id) ON DELETE CASCADE,
  added_at timestamp with time zone DEFAULT now(),
  notes text,
  UNIQUE(user_id, product_id)
);

-- User preferences table
CREATE TABLE user_preferences (
  id bigserial PRIMARY KEY,
  user_id uuid REFERENCES auth.users(id) ON DELETE CASCADE,
  preferences jsonb DEFAULT '{}',
  updated_at timestamp with time zone DEFAULT now()
);

-- Browsing history table
CREATE TABLE browsing_history (
  id bigserial PRIMARY KEY,
  user_id uuid REFERENCES auth.users(id) ON DELETE CASCADE,
  product_id bigint REFERENCES products(id) ON DELETE CASCADE,
  viewed_at timestamp with time zone DEFAULT now(),
  session_id text,
  time_spent integer DEFAULT 0, -- seconds
  UNIQUE(user_id, product_id, DATE(viewed_at))
);

-- Create indexes
CREATE INDEX idx_user_interactions_user_id ON user_interactions(user_id);
CREATE INDEX idx_user_interactions_product_id ON user_interactions(product_id);
CREATE INDEX idx_user_interactions_type ON user_interactions(interaction_type);
CREATE INDEX idx_wishlists_user_id ON wishlists(user_id);
CREATE INDEX idx_browsing_history_user_id ON browsing_history(user_id);
CREATE INDEX idx_browsing_history_viewed_at ON browsing_history(viewed_at);

-- RLS policies
ALTER TABLE user_interactions ENABLE ROW LEVEL SECURITY;
ALTER TABLE wishlists ENABLE ROW LEVEL SECURITY;
ALTER TABLE user_preferences ENABLE ROW LEVEL SECURITY;
ALTER TABLE browsing_history ENABLE ROW LEVEL SECURITY;

-- Policies for user_interactions
CREATE POLICY "Users can view own interactions" ON user_interactions
  FOR SELECT USING (auth.uid() = user_id);
CREATE POLICY "Users can insert own interactions" ON user_interactions
  FOR INSERT WITH CHECK (auth.uid() = user_id);

-- Policies for wishlists
```

```
CREATE POLICY "Users can manage own wishlist" ON wishlists
  FOR ALL USING (auth.uid() = user_id);

-- Policies for user_preferences
CREATE POLICY "Users can manage own preferences" ON user_preferences
  FOR ALL USING (auth.uid() = user_id);

-- Policies for browsing_history
CREATE POLICY "Users can view own browsing history" ON browsing_history
  FOR SELECT USING (auth.uid() = user_id);
CREATE POLICY "Users can insert own browsing history" ON browsing_history
  FOR INSERT WITH CHECK (auth.uid() = user_id);
```

Implementation Steps

Step 1: Create Personalization Context

Create `/contexts/PersonalizationContext.tsx`:

```
'use client';
import { createContext, useContext, useEffect, useState, ReactNode } from 'react';
import { useSession } from 'next-auth/react';
import { createClient } from '@/lib/supabase/client';

interface PersonalizationData {
  wishlist: WishlistItem[];
  browsingHistory: BrowsingHistoryItem[];
  preferences: UserPreferences;
  recommendations: Product[];
}

interface WishlistItem {
  id: number;
  product_id: number;
  added_at: string;
  notes?: string;
  product: Product;
}

interface BrowsingHistoryItem {
  id: number;
  product_id: number;
  viewed_at: string;
  time_spent: number;
  product: Product;
}

interface UserPreferences {
  categories: string[];
  priceRange: { min: number; max: number };
  brands: string[];
  notifications: {
    priceDrops: boolean;
    newArrivals: boolean;
    recommendations: boolean;
  };
}

interface PersonalizationContextType {
  data: PersonalizationData;
  loading: boolean;
  addToWishlist: (productId: number, notes?: string) => Promise<void>;
  removeFromWishlist: (productId: number) => Promise<void>;
  trackProductView: (productId: number, timeSpent?: number) => Promise<void>;
  updatePreferences: (preferences: Partial<UserPreferences>) => Promise<void>;
  refreshRecommendations: () => Promise<void>;
}

const PersonalizationContext = createContext<PersonalizationContextType | undefined>(un
defined);

export function PersonalizationProvider({ children }: { children: ReactNode }) {
  const { data: session } = useSession();
  const [data, setData] = useState<PersonalizationData>({
    wishlist: [],
    browsingHistory: [],
    preferences: {
      categories: [],
      priceRange: { min: 0, max: 1000 },
      brands: [],
      notifications: {
```

```
      priceDrops: true,
      newArrivals: true,
      recommendations: true,
    },
  },
  recommendations: [],
});
const [loading, setLoading] = useState(true);

const supabase = createClient();

useEffect(() => {
  if (session?.user?.id) {
    loadPersonalizationData();
  } else {
    setLoading(false);
  }
}, [session?.user?.id]);

const loadPersonalizationData = async () => {
  if (!session?.user?.id) return;

  try {
    setLoading(true);

    // Load wishlist
    const { data: wishlistData } = await supabase
      .from('wishlists')
      .select(`
        *,
        product:products(*)
      `)
      .eq('user_id', session.user.id)
      .order('added_at', { ascending: false });

    // Load browsing history
    const { data: historyData } = await supabase
      .from('browsing_history')
      .select(`
        *,
        product:products(*)
      `)
      .eq('user_id', session.user.id)
      .order('viewed_at', { ascending: false })
      .limit(50);

    // Load preferences
    const { data: preferencesData } = await supabase
      .from('user_preferences')
      .select('preferences')
      .eq('user_id', session.user.id)
      .single();

    // Load recommendations
    const recommendationsResponse = await fetch('/api/recommendations/personalized');
    const recommendationsData = await recommendationsResponse.json();

    setData({
      wishlist: wishlistData || [],
      browsingHistory: historyData || [],
      preferences: preferencesData?.preferences || data.preferences,
      recommendations: recommendationsData.products || [],
    });
```

```
    } catch (error) {
      console.error('Error loading personalization data:', error);
    } finally {
      setLoading(false);
    }
  };

  const addToWishlist = async (productId: number, notes?: string) => {
    if (!session?.user?.id) return;

    try {
      const { error } = await supabase
        .from('wishlists')
        .insert({
          user_id: session.user.id,
          product_id: productId,
          notes,
        });

      if (error) throw error;

      // Track interaction
      await trackInteraction(productId, 'wishlist_add');

      // Reload wishlist
      await loadPersonalizationData();
    } catch (error) {
      console.error('Error adding to wishlist:', error);
    }
  };

  const removeFromWishlist = async (productId: number) => {
    if (!session?.user?.id) return;

    try {
      const { error } = await supabase
        .from('wishlists')
        .delete()
        .eq('user_id', session.user.id)
        .eq('product_id', productId);

      if (error) throw error;

      // Track interaction
      await trackInteraction(productId, 'wishlist_remove');

      // Reload wishlist
      await loadPersonalizationData();
    } catch (error) {
      console.error('Error removing from wishlist:', error);
    }
  };

  const trackProductView = async (productId: number, timeSpent: number = 0) => {
    if (!session?.user?.id) return;

    try {
      // Update or insert browsing history
      const { error } = await supabase
        .from('browsing_history')
        .upsert({
          user_id: session.user.id,
          product_id: productId,
```

```
          time_spent: timeSpent,
          viewed_at: new Date().toISOString(),
        });

      if (error) throw error;

      // Track interaction
      await trackInteraction(productId, 'view', { time_spent: timeSpent });
    } catch (error) {
      console.error('Error tracking product view:', error);
    }
  };

  const trackInteraction = async (
    productId: number,
    type: string,
    metadata: any = {}
  ) => {
    if (!session?.user?.id) return;

    try {
      await supabase
        .from('user_interactions')
        .insert({
          user_id: session.user.id,
          product_id: productId,
          interaction_type: type,
          metadata,
        });
    } catch (error) {
      console.error('Error tracking interaction:', error);
    }
  };

  const updatePreferences = async (newPreferences: Partial<UserPreferences>) => {
    if (!session?.user?.id) return;

    try {
      const updatedPreferences = { ...data.preferences, ...newPreferences };

      const { error } = await supabase
        .from('user_preferences')
        .upsert({
          user_id: session.user.id,
          preferences: updatedPreferences,
        });

      if (error) throw error;

      setData(prev => ({
        ...prev,
        preferences: updatedPreferences,
      }));
    } catch (error) {
      console.error('Error updating preferences:', error);
    }
  };

  const refreshRecommendations = async () => {
    try {
      const response = await fetch('/api/recommendations/personalized');
      const data = await response.json();
```

```
      setData(prev => ({
        ...prev,
        recommendations: data.products || [],
      }));
    } catch (error) {
      console.error('Error refreshing recommendations:', error);
    }
  };

  return (
    <PersonalizationContext.Provider
      value={{
        data,
        loading,
        addToWishlist,
        removeFromWishlist,
        trackProductView,
        updatePreferences,
        refreshRecommendations,
      }}
    >
      {children}
    </PersonalizationContext.Provider>
  );
}

export function usePersonalization() {
  const context = useContext(PersonalizationContext);
  if (context === undefined) {
    throw new Error('usePersonalization must be used within a PersonalizationProvider')
;
  }
  return context;
}
```

Step 2: Create Wishlist Component

Create `/components/wishlist/WishlistButton.tsx`:

```
'use client';
import { useState } from 'react';
import { Heart } from 'lucide-react';
import { Button } from '@/components/ui/button';
import { usePersonalization } from '@/contexts/PersonalizationContext';
import { useSession } from 'next-auth/react';
import { toast } from 'sonner';

interface WishlistButtonProps {
  productId: number;
  className?: string;
}

export default function WishlistButton({ productId, className = '' }: WishlistButton-
Props) {
  const { data: session } = useSession();
  const { data, addToWishlist, removeFromWishlist } = usePersonalization();
  const [loading, setLoading] = useState(false);

  const isInWishlist = data.wishlist.some(item => item.product_id === productId);

  const handleToggleWishlist = async () => {
    if (!session) {
      toast.error('Please sign in to use wishlist');
      return;
    }

    setLoading(true);
    try {
      if (isInWishlist) {
        await removeFromWishlist(productId);
        toast.success('Removed from wishlist');
      } else {
        await addToWishlist(productId);
        toast.success('Added to wishlist');
      }
    } catch (error) {
      toast.error('Failed to update wishlist');
    } finally {
      setLoading(false);
    }
  };

  return (
    <Button
      variant={isInWishlist ? 'default' : 'outline'}
      size="sm"
      onClick={handleToggleWishlist}
      disabled={loading}
      className={className}
    >
      <Heart
        className={`w-4 h-4 mr-1 ${isInWishlist ? 'fill-current' : ''}`}
      />
      {isInWishlist ? 'In Wishlist' : 'Add to Wishlist'}
    </Button>
  );
}
```

Create `/app/wishlist/page.tsx`:

```
'use client';
import { usePersonalization } from '@/contexts/PersonalizationContext';
import { Button } from '@/components/ui/button';
import { Card, CardContent, CardHeader, CardTitle } from '@/components/ui/card';
import { Trash2, ShoppingCart } from 'lucide-react';
import Link from 'next/link';
import OptimizedImage from '@/components/OptimizedImage';

export default function WishlistPage() {
  const { data, loading, removeFromWishlist } = usePersonalization();

  if (loading) {
    return (
      <div className="container mx-auto px-4 py-8">
        <div className="animate-pulse space-y-4">
          {[...Array(3)].map((_, i) => (
            <div key={i} className="h-32 bg-gray-200 rounded-lg" />
          ))}
        </div>
      </div>
    );
  }

  if (data.wishlist.length === 0) {
    return (
      <div className="container mx-auto px-4 py-8 text-center">
        <h1 className="text-3xl font-bold mb-4">Your Wishlist</h1>
        <p className="text-gray-600 mb-8">Your wishlist is empty</p>
        <Link href="/products">
          <Button>Browse Products</Button>
        </Link>
      </div>
    );
  }

  return (
    <div className="container mx-auto px-4 py-8">
      <h1 className="text-3xl font-bold mb-8">Your Wishlist ({data.wishlist.length} ite
ms)</h1>

      <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-6">
        {data.wishlist.map((item) => (
          <Card key={item.id} className="overflow-hidden">
            <div className="relative">
              <OptimizedImage
                src={item.product.image_url}
                alt={item.product.title}
                width={300}
                height={200}
                className="w-full h-48 object-cover"
              />
            </div>

            <CardHeader>
              <CardTitle className="line-clamp-2">{item.product.title}</CardTitle>
              <p className="text-2xl font-bold text-green-600">${item.product.price}</
p>
            </CardHeader>

            <CardContent>
              <p className="text-gray-600 text-sm mb-4 line-clamp-3">
                {item.product.description}
```

```
                </p>

                {item.notes && (
                  <p className="text-sm text-blue-600 mb-4 italic">
                    Note: {item.notes}
                  </p>
                )}

                <div className="flex gap-2">
                  <Button asChild className="flex-1">
                    <Link href={`/products/${item.product.id}`}>
                      <ShoppingCart className="w-4 h-4 mr-1" />
                      View Product
                    </Link>
                  </Button>

                  <Button
                    variant="outline"
                    size="sm"
                    onClick={() => removeFromWishlist(item.product_id)}
                  >
                    <Trash2 className="w-4 h-4" />
                  </Button>
                </div>

                <p className="text-xs text-gray-500 mt-2">
                  Added {new Date(item.added_at).toLocaleDateString()}
                </p>
              </CardContent>
            </Card>
          ))}
        </div>
      </div>
    );
}
```

Step 3: Create Browsing History Component

Create `/components/history/BrowsingHistory.tsx`:

```
'use client';
import { usePersonalization } from '@/contexts/PersonalizationContext';
import { Card, CardContent, CardHeader, CardTitle } from '@/components/ui/card';
import { Button } from '@/components/ui/button';
import { Clock, Eye } from 'lucide-react';
import Link from 'next/link';
import OptimizedImage from '@/components/OptimizedImage';

export default function BrowsingHistory() {
  const { data, loading } = usePersonalization();

  if (loading) {
    return <div className="animate-pulse h-64 bg-gray-200 rounded-lg" />;
  }

  if (data.browsingHistory.length === 0) {
    return (
      <Card>
        <CardHeader>
          <CardTitle className="flex items-center gap-2">
            <Clock className="w-5 h-5" />
            Recently Viewed
          </CardTitle>
        </CardHeader>
        <CardContent>
          <p className="text-gray-600">No browsing history yet</p>
        </CardContent>
      </Card>
    );
  }

  return (
    <Card>
      <CardHeader>
        <CardTitle className="flex items-center gap-2">
          <Clock className="w-5 h-5" />
          Recently Viewed ({data.browsingHistory.length})
        </CardTitle>
      </CardHeader>
      <CardContent>
        <div className="space-y-4">
          {data.browsingHistory.slice(0, 5).map((item) => (
            <div key={item.id}
className="flex items-center gap-4 p-3 border rounded-lg hover:bg-gray-50">
              <OptimizedImage
                src={item.product.image_url}
                alt={item.product.title}
                width={60}
                height={60}
                className="w-15 h-15 object-cover rounded"
              />

              <div className="flex-1 min-w-0">
                <h4 className="font-medium line-clamp-1">{item.product.title}</h4>
                <p className="text-sm text-gray-600">${item.product.price}</p>
                <div className="flex items-center gap-2 text-xs text-gray-500 mt-1">
                  <Eye className="w-3 h-3" />
                  <span>{new Date(item.viewed_at).toLocaleDateString()}</span>
                  {item.time_spent > 0 && (
                    <span>• {Math.round(item.time_spent / 60)}m viewed</span>
                  )}
                </div>
```

```
              </div>

              <Button asChild size="sm" variant="outline">
                <Link href={`/products/${item.product.id}`}>
                  View
                </Link>
              </Button>
            </div>
          ))}

          {data.browsingHistory.length > 5 && (
            <Button asChild variant="ghost" className="w-full">
              <Link href="/history">View All History</Link>
            </Button>
          )}
        </div>
      </CardContent>
    </Card>
  );
}
```

Step 4: Create Personalized Recommendations API

Create `/app/api/recommendations/personalized/route.ts`:

```
import { NextRequest, NextResponse } from 'next/server';
import { auth } from '@/auth';
import { getRecommendations } from '@/lib/recommendations';

export async function GET(request: NextRequest) {
  try {
    const session = await auth();

    if (!session?.user?.id) {
      return NextResponse.json({ error: 'Unauthorized' }, { status: 401 });
    }

    const { searchParams } = new URL(request.url);
    const limit = parseInt(searchParams.get('limit') || '10');
    const type = searchParams.get('type') || 'personalized';

    const recommendations = await getRecommendations({
      userId: session.user.id,
      limit,
      type: type as any,
    });

    return NextResponse.json({
      products: recommendations,
      userId: session.user.id,
      type,
    });
  } catch (error) {
    console.error('Recommendations error:', error);
    return NextResponse.json(
      { error: 'Failed to get recommendations' },
      { status: 500 }
    );
  }
}
```

Step 5: Create Product View Tracking Hook

Create `/hooks/useProductTracking.ts`:

```
'use client';
import { useEffect, useRef } from 'react';
import { usePersonalization } from '@/contexts/PersonalizationContext';
import { useSession } from 'next-auth/react';

export function useProductTracking(productId: number) {
  const { data: session } = useSession();
  const { trackProductView } = usePersonalization();
  const startTimeRef = useRef<number>(Date.now());
  const trackedRef = useRef<boolean>(false);

  useEffect(() => {
    if (!session?.user?.id || trackedRef.current) return;

    // Track initial view
    trackProductView(productId);
    trackedRef.current = true;
    startTimeRef.current = Date.now();

    // Track time spent when component unmounts
    return () => {
      const timeSpent = Math.round((Date.now() - startTimeRef.current) / 1000);
      if (timeSpent > 5) { // Only track if spent more than 5 seconds
        trackProductView(productId, timeSpent);
      }
    };
  }, [productId, session?.user?.id, trackProductView]);

  // Track visibility changes
  useEffect(() => {
    const handleVisibilityChange = () => {
      if (document.hidden && session?.user?.id) {
        const timeSpent = Math.round((Date.now() - startTimeRef.current) / 1000);
        if (timeSpent > 5) {
          trackProductView(productId, timeSpent);
        }
      } else if (!document.hidden) {
        startTimeRef.current = Date.now();
      }
    };

    document.addEventListener('visibilitychange', handleVisibilityChange);
    return () => document.removeEventListener('visibilitychange', handleVisibili-
tyChange);
  }, [productId, session?.user?.id, trackProductView]);
}
```

Success Criteria

- [ ] Users can add/remove items from wishlist
- [ ] Browsing history is tracked and displayed
- [ ] Personalized recommendations are accurate
- [ ] User preferences can be updated
- [ ] All interactions are properly logged
- [ ] Performance remains optimal with tracking

Testing Procedures

Comprehensive Test Plan

Unit Tests

```
# Run unit tests
npm run test

# Run with coverage
npm run test:coverage
```

E2E Tests

```
# Run E2E tests
npm run test:e2e

# Run in headed mode for debugging
npm run test:e2e:headed
```

Performance Tests

```
# Run Lighthouse CI
npx lhci autorun

# Run custom performance tests
npm run test:performance
```

Test Scenarios by Feature

Google OAuth Tests

- [ ] Login flow completes successfully
- [ ] User session persists across refreshes
- [ ] Protected routes redirect correctly
- [ ] Logout clears session
- [ ] Error handling for failed authentication

Performance Tests

- [ ] Page load times under 2 seconds
- [ ] Core Web Vitals in “Good” range
- [ ] Build times improved with Turbopack
- [ ] PPR working for static/dynamic content
- [ ] Background tasks don’t block responses

Search & Discovery Tests

- [ ] Semantic search returns relevant results
- [ ] Keyword search works correctly
- [ ] Hybrid search combines both effectively
- [ ] Filters apply correctly
- [ ] Recommendations are personalized

Personalization Tests

- [ ] Wishlist add/remove functionality
- [ ] Browsing history tracking
- [ ] Personalized recommendations
- [ ] User preferences persistence
- [ ] Interaction logging

Potential Issues & Troubleshooting

Common Issues

Google OAuth Issues

Problem: OAuth callback fails
Solution:
- Check redirect URIs in Google Console
- Verify environment variables
- Ensure AUTH_SECRET is set

Performance Issues

Problem: Slow page loads
Solution:
- Enable Turbopack in development
- Check for unoptimized images
- Review database query performance

Search Issues

Problem: Semantic search not working
Solution:
- Verify pgvector extension is enabled
- Check embedding generation
- Validate vector dimensions match

Personalization Issues

Problem: Recommendations not updating
Solution:
- Check user interaction logging
- Verify recommendation algorithm
- Clear cache if needed

Debugging Tools

Development Tools

```
# Enable debug mode
DEBUG=* npm run dev

# Check database connections
npm run db:check

# Analyze bundle size
npm run analyze
```

Production Monitoring

- Set up error tracking (Sentry)
- Monitor performance (Vercel Analytics)
- Track user interactions (PostHog)

Configuration Details

Environment Variables

```
# Authentication
AUTH_SECRET=your-secret-key
AUTH_GOOGLE_ID=your-google-client-id
AUTH_GOOGLE_SECRET=your-google-client-secret
AUTH_TRUST_HOST=true

# Supabase
NEXT_PUBLIC_SUPABASE_URL=your-supabase-url
NEXT_PUBLIC_SUPABASE_ANON_KEY=your-anon-key
SUPABASE_SERVICE_ROLE_KEY=your-service-role-key

# Stripe
STRIPE_SECRET_KEY=your-stripe-secret
STRIPE_WEBHOOK_SECRET=your-webhook-secret
NEXT_PUBLIC_STRIPE_PUBLISHABLE_KEY=your-publishable-key

# Site
NEXT_PUBLIC_SITE_URL=https://your-domain.com
```

Deployment Configuration

Netlify Settings

```
# netlify.toml
[build]
  command = "npm run build"
  publish = ".next"

[build.environment]
  NODE_VERSION = "18"

[[redirects]]
  from = "/api/*"
  to = "/.netlify/functions/:splat"
  status = 200
```

Timeline & Milestones

Feature 1: Google OAuth (Week 1-2)

- **Week 1**: Setup and configuration
- [ ] Install NextAuth v5
- [ ] Configure Google OAuth
- [ ] Create auth pages
- [ ] Set up middleware
- **Week 2**: Testing and refinement
- [ ] Write tests
- [ ] Handle edge cases
- [ ] Update admin integration
- [ ] Deploy and verify

Feature 2: Performance Optimization (Week 2-3)

- **Week 2**: Core optimizations (parallel with OAuth)
- [ ] Enable PPR
- [ ] Configure Turbopack
- [ ] Optimize caching
- **Week 3**: Advanced features
- [ ] Implement background tasks
- [ ] Add performance monitoring
- [ ] Optimize images and assets
- [ ] Performance testing

Feature 3: Enhanced Product Discovery (Week 4-6)

- **Week 4**: Database and embeddings
- [ ] Set up pgvector
- [ ] Generate embeddings
- [ ] Create search functions

- **Week 5**: Search implementation
- [ ] Build search API
- [ ] Create search components
- [ ] Implement filtering
- **Week 6**: Recommendations
- [ ] Build recommendation engine
- [ ] Create recommendation components
- [ ] Testing and optimization

Feature 4: Personalized Shopping (Week 6-8)

- **Week 6**: Foundation (parallel with discovery)
- [ ] Database schema
- [ ] Personalization context
- [ ] Basic tracking
- **Week 7**: Core features
- [ ] Wishlist functionality
- [ ] Browsing history
- [ ] User preferences
- **Week 8**: Integration and testing
- [ ] Personalized recommendations
- [ ] Complete integration
- [ ] Comprehensive testing
- [ ] Performance optimization

Final Integration (Week 8-9)

- [ ] End-to-end testing
- [ ] Performance optimization
- [ ] Documentation updates
- [ ] Production deployment
- [ ] Monitoring setup

Success Metrics

Performance Metrics

- Page load time: < 2 seconds
- Core Web Vitals: All “Good”
- Build time improvement: 40%+
- Search response time: < 500ms

User Experience Metrics

- OAuth conversion rate: > 80%
- Search relevance score: > 85%
- Recommendation click-through: > 15%
- Wishlist usage: > 30% of users

Technical Metrics

- Test coverage: > 90%
- Zero critical security issues
- 99.9% uptime
- Error rate: < 0.1%

Next Steps for Implementation

1. **Start with Google OAuth** - Foundation for personalization
2. **Implement Performance Optimizations** - Can run parallel
3. **Build Enhanced Search** - Requires performance foundation
4. **Add Personalization** - Requires OAuth and search
5. **Integration Testing** - Ensure all features work together
6. **Performance Monitoring** - Set up comprehensive monitoring
7. **User Feedback** - Gather feedback and iterate

This handover document provides everything needed to implement the 4 high-impact features successfully. Each section includes detailed technical specifications, step-by-step implementation guides, testing procedures, and troubleshooting information.