# Complete Project Setup Guide - Wine Store

## 1. Initial Project Setup
```bash
# Create Next.js project with TypeScript and App Router
npx create-next-app@latest store-wine
cd store-wine
```

## 2. Install All Dependencies
```bash
# Install Prisma
npm install prisma --save-dev
npm install @prisma/client

# Install Supabase dependencies
npm install @supabase/supabase-js
npm install @supabase/auth-helpers-nextjs
npm install @supabase/ssr

# Install UI and styling dependencies
npm install @radix-ui/react-dropdown-menu
npm install @radix-ui/react-slot
npm install @radix-ui/react-dialog
npm install class-variance-authority
npm install clsx
npm install lucide-react
npm install next-themes
npm install tailwind-merge
npm install tailwindcss-animate

# Install form handling
npm install react-hook-form
npm install @hookform/resolvers
npm install zod

# Install additional utilities
npm install axios
npm install date-fns

# Install Supabase CLI globally
npm install -g supabase
```

## 3. Supabase Setup
```bash
# Initialize Supabase CLI
supabase init

# Login to Supabase CLI
supabase login

# Start Supabase locally
supabase start

# Or create new project via dashboard: https://app.supabase.com/projects
```

## 4. Environment Setup
```bash
# Create .env file
touch .env

# Add these variables to .env:
DATABASE_URL="postgresql://postgres:[YOUR-PASSWORD]@db.[YOUR-PROJECT-REF].supabase.co:5432/postgres"
NEXT_PUBLIC_SITE_URL="http://localhost:3000"
NEXT_PUBLIC_SUPABASE_URL="your-project-url"
NEXT_PUBLIC_SUPABASE_ANON_KEY="your-anon-key"
SUPABASE_SERVICE_ROLE_KEY="your-service-role-key"
```

## 5. Initialize Prisma with Supabase
```bash
# Initialize Prisma
npx prisma init

# Pull the schema from Supabase
npx prisma db pull

# Generate Prisma Client
npx prisma generate

# Push schema to database
```

~~~
npx prisma db push
```

## 6. Setup Shadcn UI
```bash
# Initialize Shadcn UI
npx shadcn-ui@latest init
# Add required components
npx shadcn-ui@latest add button
npx shadcn-ui@latest add dropdown-menu
npx shadcn-ui@latest add input
npx shadcn-ui@latest add dialog
npx shadcn-ui@latest add form
npx shadcn-ui@latest add card
```

## 7. Set up Supabase Client
```typescript
// utils/supabase.ts
import { createClient } from '@supabase/supabase-js'

const supabaseUrl = process.env.NEXT_PUBLIC_SUPABASE_URL!
const supabaseKey = process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY!

export const supabase = createClient(supabaseUrl, supabaseKey)
```

## 8. Set up Middleware
```typescript
// middleware.ts
import { createMiddlewareClient } from '@supabase/auth-helpers-nextjs'
import { NextResponse } from 'next/server'
import type { NextRequest } from 'next/server'

export async function middleware(req: NextRequest) {
 const res = NextResponse.next()
 const supabase = createMiddlewareClient({ req, res })
 await supabase.auth.getSession()
 return res
}
```

## 9. Database Security Setup
```sql
-- Enable RLS
alter table "public"."wines" enable row level security;

-- Create policies
create policy "Public wines are viewable by everyone"
 on wines for select
 using ( true );

create policy "Users can insert their own wines"
 on wines for insert
 with check ( auth.uid() = user_id );
```

## 10. Development Commands
```bash
# Run development server
npm run dev

# Build for production
npm run build

# Start production server
npm run start

# Prisma commands
npx prisma studio
npx prisma generate
npx prisma db push

# Supabase commands
supabase db pull
supabase db push
supabase gen types typescript --local > types/supabase.ts
supabase logs
supabase db reset
supabase studio
```

## 11. Git Setup
```bash
~~~

```
git init
git add .
git commit -m "Initial commit"
```

## 12. Required VSCode Extensions
- Prisma (prisma.prisma)
- Tailwind CSS IntelliSense (bradlc.vscode-tailwindcss)
- ESLint (dbaeumer.vscode-eslint)
- TypeScript ESLint (dbaeumer.vscode-eslint)

## 13. System Requirements
- Node.js 18.17 or later
- npm version 9+
- PostgreSQL installed locally
- Ports 3000 (Next.js) and 5432 (PostgreSQL) available

## Important Notes:
1. Always backup your database before running migrations
2. Test migrations locally before deploying to production
3. Keep environment variables secure and never commit them
4. Use Row Level Security (RLS) policies for data protection
5. Keep Prisma schema and Supabase schema synchronized
6. Generate types for better TypeScript support
7. Use Supabase's built-in authentication system
8. Follow Next.js App Router best practices
9. Implement proper error boundaries
10. Use appropriate loading and error states
11. Maintain component modularity
12. Follow TypeScript best practices
13. Use proper security headers
14. Implement proper data sanitization
15. Use proper session management

## File Structure

```
store-wine/
├── .next/
├── app/
│   ├── about/
│   ├── admin/
│   ├── cart/
│   ├── favorites/
│   ├── orders/
│   ├── products/
│   ├── reviews/
│   ├── favicon.ico
│   ├── globals.css
│   ├── globalcss.back
│   ├── layout.tsx
│   ├── page.tsx
│   ├── providers.tsx
│   └── theme-provider.tsx
├── components/
│   ├── cart/
│   ├── form/
│   ├── global/
│   ├── home/
│   ├── navbar/
│   ├── products/
│   ├── single-product/
│   └── ui/
├── lib/
├── prisma/
├── public/
├── utils/
├── .DS_Store
├── .gitignore
├── BDCC-07-00020-v2.pdf
├── README.md
├── Tutorial.md
├── Tutorial.pdf
├── back-schema.prisma0
├── backup schema
├── components.json
├── eslint.config.mjs
├── next-env.d.ts
├── next.config.ts
├── package-lock.json
├── package.json
├── postcss.config.mjs
└── tsconfig.json
```

# Wine Store Frontend Documentation

## Project Overview
This is a Next.js-based wine store application using the App Router pattern. The project implements a modern, responsive design with dark mode support and a component-based architecture.

## Core Technical Stack
- Next.js (App Router)
- TypeScript
- Tailwind CSS
- Next.js built-in components and routing

## Project Structure

### Root Layout (`app/layout.tsx`)
The application uses a root layout that provides:
- Google Inter font integration
- Global CSS
- Theme provider integration
- Responsive container layout
- Global navigation

### Theme Management (`app/providers.tsx`, `app/theme-provider.tsx`)
- Implements dark mode support using Next.js theme provider
- Supports system theme detection
- Provides smooth theme transitions
- Client-side theme switching capability

### Navigation (`components/navbar/Navbar.tsx`)
The navigation bar includes:
- Logo component
- Search functionality
- Shopping cart access
- Dark mode toggle
- Dropdown menu for additional links
- Responsive design with mobile optimization

### Core Pages
1. Products Page (`app/products/page.tsx`)
  - Basic structure implemented
  - Ready for product listing implementation

2. Cart Page (`app/cart/page.tsx`)
  - Basic structure implemented
  - Ready for cart functionality implementation

### Component Organization
```
components/
├── cart/
├── form/
├── global/
│   └── Container.tsx    # Reusable container component
├── home/
├── navbar/
│   ├── Navbar.tsx      # Main navigation component
│   ├── Logo.tsx
│   ├── LinksDropdown.tsx
│   ├── DarkMode.tsx
│   ├── CartButton.tsx
│   └── NavSearch.tsx
├── products/
├── single-product/
└── ui/
```

### Global Components
1. Container Component
  - Provides consistent padding and max-width
  - Supports custom className props for flexibility

2. Navbar Components
  - Modular design with separate components for each function
  - Responsive layout with mobile-first approach
  - Integrated dark mode toggle
  - Search functionality
  - Cart integration

## Routing Structure
```
app/
├── about/
├── admin/
```

~~~markdown
├── cart/
├── favorites/
├── orders/
├── products/
└── reviews/
```

## Current Implementation Status
The project has established its core architecture with:
- ✅ Basic routing structure
- ✅ Theme management
- ✅ Navigation framework
- ✅ Responsive layout system
- ✅ Component organization

Pending Implementation:
- 🚧 Product listing and filtering
- 🚧 Shopping cart functionality
- 🚧 User authentication
- 🚧 Order management
- 🚧 Review system
- 🚧 Admin interface

## Development Guidelines

### 1. Component Structure
- Use TypeScript for all components
- Implement client components with "use client" directive where needed
- Follow the established component organization pattern

### 2. Styling
- Use Tailwind CSS for styling
- Follow the established class naming conventions
- Utilize the Container component for consistent layout

### 3. Navigation
- Add new routes in the app directory
- Update LinksDropdown component when adding new navigation items
- Maintain responsive design considerations

## Key Implementation Details

### Layout Implementation
```typescript
// app/layout.tsx
import type { Metadata } from "next";
import { Inter } from "next/font/google";
import "./globals.css";
import Providers from "./providers";
import Navbar from "@/components/navbar/Navbar";
import Container from "@/components/global/Container";

const inter = Inter({ subsets: ["latin"] });

export const metadata: Metadata = {
 title: "Wine Store",
 description: "BabyFox",
};

export default function RootLayout({
 children,
}: Readonly<{
 children: React.ReactNode;
}>) {
 return (
   <html lang="en" suppressHydrationWarning>
     <body className={inter.className}>
       <Providers>
         <Navbar />
         <Container className="py-20">{children}</Container>
       </Providers>
     </body>
   </html>
 );
}
```

// app/page.tsx
import React from "react";
import { Button } from "@/components/ui/button";
import { MailOpen } from "lucide-react";

function HomePage() {
 return (
~~~

```
   <div>
     <h1 className="text-3xl">HomePage</h1>
     <Button variant="default" size="lg" className="capitalize m-8">
       <MailOpen /> Click me
     </Button>
   </div>
 );
}

export default HomePage;


// components/navbar/CartButton.tsx
import { Button } from "@/components/ui/button";
import { LuShoppingCart } from "react-icons/lu";
import Link from "next/link";
async function CartButton() {
 const numItemsInCart = 9;
 return (
   <Button
     asChild
     variant="outline"
     size="icon"
     className="flex justify-center items-center relative"
   >
     <Link href="/cart">
       <LuShoppingCart />
       <span className="absolute -top-3 -right-3 bg-primary text-white rounded-full h-6 w-6
flex items-center justify-center text-xs">
         {numItemsInCart}
       </span>
     </Link>
   </Button>
 );
}
export default CartButton;

// components/navbar/DarkMode.tsx
"use client";
import { Button } from "@/components/ui/button";
import { Moon, Sun } from "lucide-react";
import { useTheme } from "next-themes";

function DarkMode() {
 const { theme, setTheme } = useTheme();
 return (
   <Button
     variant="outline"
     size="icon"
     onClick={() => {
       if (theme === "dark") {
         setTheme("light");
       } else {
         setTheme("dark");
       }
     }}
   >
     <Sun className="h-[1.2rem] w-[1.2rem] rotate-0 scale-100 transition-all dark:-rotate-90
dark:scale-0" />
     <Moon className="absolute h-[1.2rem] w-[1.2rem] rotate-90 scale-0 transition-all
dark:rotate-0 dark:scale-100" />
     <span className="sr-only">Toggle theme</span>
   </Button>
 );
}
export default DarkMode;

// components/navbar/LinksDropdown.tsx
import {
 DropdownMenu,
 DropdownMenuContent,
 DropdownMenuItem,
 DropdownMenuTrigger,
} from "@/components/ui/dropdown-menu";
import { Button } from "@/components/ui/button";
import { AlignJustify } from "lucide-react";
import Link from "next/link";

function LinksDropdown() {
 return (
   <DropdownMenu>
     <DropdownMenuTrigger asChild>
       <Button variant="outline" size="icon">
         <AlignJustify />
```

~~~
        </Button>
      </DropdownMenuTrigger>
      <DropdownMenuContent align="end">
        <DropdownMenuItem asChild>
          <Link href="/about">About</Link>
        </DropdownMenuItem>
        <DropdownMenuItem asChild>
          <Link href="/products">Products</Link>
        </DropdownMenuItem>
        <DropdownMenuItem asChild>
          <Link href="/cart">Cart</Link>
        </DropdownMenuItem>
      </DropdownMenuContent>
    </DropdownMenu>
  );
}
export default LinksDropdown;

// components/navbar/Logo.tsx
import { Button } from "@/components/ui/button";
import { GiWineBottle } from "react-icons/gi";
import Link from "next/link";

function Logo() {
  return (
    <Button asChild variant="link" className="!text-2xl">
      <Link href="/" className="flex items-center gap-2">
        <GiWineBottle className="text-primary" />
        <span className="tracking-wider">
          Wine<span className="text-primary">Store</span>
        </span>
      </Link>
    </Button>
  );
}
export default Logo;

// components/navbar/NavSearch.tsx
import { Input } from "@/components/ui/input";

function NavSearch() {
  return (
    <Input type="search" placeholder="Search" className="w-full sm:w-[300px]" />
  );
}
export default NavSearch;


### Theme Provider Implementation
```typescript
// app/providers.tsx
"use client";
import { ThemeProvider } from "./theme-provider";

function Providers({ children }: { children: React.ReactNode }) {
  return (
    <ThemeProvider
      attribute="class"
      defaultTheme="system"
      enableSystem
      disableTransitionOnChange
    >
      {children}
    </ThemeProvider>
  );
}
export default Providers;
```

### Navigation Implementation
```typescript
// components/navbar/Navbar.tsx
import Logo from "./Logo";
import LinksDropdown from "./LinksDropdown";
import DarkMode from "./DarkMode";
import CartButton from "./CartButton";
import NavSearch from "./NavSearch";
import Container from "../global/Container";

function Navbar() {
  return (
    <nav className="border-b">
      <Container className="flex flex-col sm:flex-row sm:justify-between sm:items-center
flex-wrap gap-4 py-8">
~~~

~~~
       <Logo />
       <NavSearch />
       <div className="flex gap-4 items-center">
         <CartButton />
         <DarkMode />
         <LinksDropdown />
       </div>
     </Container>
   </nav>
 );
}
export default Navbar;
```

## Best Practices
1. Always use TypeScript for type safety
2. Implement proper error boundaries
3. Use appropriate loading and error states
4. Follow Next.js App Router conventions
5. Maintain component modularity
6. Use Tailwind CSS utility classes consistently
7. Implement responsive design from the start
8. Keep components small and focused
9. Use proper TypeScript types for props
10. Maintain consistent file and folder naming conventions

## Performance Considerations
1. Implement proper code splitting
2. Use Next.js Image component for optimized images
3. Implement proper caching strategies
4. Use server components where appropriate
5. Optimize client-side bundle size
6. Implement proper loading states
7. Use proper meta tags for SEO
8. Implement proper error handling
9. Use proper data fetching strategies
10. Implement proper state management

## Security Considerations
1. Implement proper authentication
2. Use proper CSRF protection
3. Implement proper input validation
4. Use proper error handling
5. Implement proper access control
6. Use proper security headers
7. Implement proper data sanitization
8. Use proper session management
9. Implement proper logging
10. Use proper environment variables




# Wine Store Database Implementation Guide - Backend
Last Updated: March 19, 2024

## 1. Schema Definition (prisma/schema.prisma)
```prisma
generator client {
 provider = "prisma-client-js"
}

datasource db {
 provider = "postgresql"
 url      = env("DATABASE_URL")
 directUrl = env("DIRECT_URL")
}

model Wine {
 id        Int      @id @default(autoincrement())
 name      String
 type      String
 elaborate String?
 grapes    String
 harmonize String
 abv       Float
 body      String
 acidity   String
 code      String
 price     Int      @default(0)
 regionId  Int
 region    Region   @relation(fields: [regionId], references: [id])
 ratings   Rating[]
 images    Image[]
~~~

```
}

model Region {
 id      Int     @id @default(autoincrement())
 name    String
 country String
 wines   Wine[]
}

model Rating {
 id      Int      @id @default(autoincrement())
 wineId  Int
 userId  Int
 vintage String?
 rating  Float
 date    DateTime
 wine    Wine     @relation(fields: [wineId], references: [id])
 user    User     @relation(fields: [userId], references: [id])
}

model User {
 id       Int      @id @default(autoincrement())
 email    String   @unique
 name     String?
 password String
 role     String   @default("USER")
 ratings  Rating[]
}

model Image {
 id     Int    @id @default(autoincrement())
 url    String
 wineId Int
 wine   Wine   @relation(fields: [wineId], references: [id])
}
```

## 2. Environment Setup
```env
# .env
DATABASE_URL="postgres://postgres.[PROJECT-REF]:[PASSWORD]@aws-0-[REGION].pooler.supabase.com:6543/postgres"
DIRECT_URL="postgresql://postgres:[PASSWORD]@db.[PROJECT-REF].supabase.com:5432/postgres"
SUPABASE_URL="https://[PROJECT-REF].supabase.co"
SUPABASE_ANON_KEY="your-anon-key"
SUPABASE_SERVICE_ROLE_KEY="your-service-role-key"
DB_POOL_MIN=2
DB_POOL_MAX=10
DB_POOL_IDLE_TIMEOUT=30000
BACKUP_PATH="./backups"
BACKUP_RETENTION_DAYS=30
```

## 3. Database Initialization
```bash
# Generate Prisma client
npx prisma generate

# Create migration
npx prisma migrate dev --name initial_schema

# Push schema to database
npx prisma db push
```

## 4. Data Seeding (prisma/seed.js)
```javascript
const { PrismaClient } = require('@prisma/client')
const { parse } = require('csv-parse')
const fs = require('fs')

const prisma = new PrismaClient()

async function main() {
 // Create regions
 const regions = [
   { name: "Bordeaux", country: "France" },
   { name: "Tuscany", country: "Italy" },
   { name: "Napa Valley", country: "USA" }
 ]
  for (const region of regions) {
   await prisma.region.create({ data: region })
 }
 // Import wines
 const wines = fs.readFileSync('prisma/wine100.csv')
 parse(wines, {
   columns: true,
   skip_empty_lines: true
 }, async (err, records) => {
   for (const record of records) {
     await prisma.wine.create({
       data: {
         name: record.name,
         type: record.type,
         grapes: record.grapes,
         harmonize: record.harmonize,
         abv: parseFloat(record.abv),
         body: record.body,
         acidity: record.acidity,
         code: record.code,
         price: parseInt(record.price),
         regionId: parseInt(record.regionId)
       }
     })
   }
 })

 // Import ratings
 const ratings = fs.readFileSync('prisma/rating1000.csv')
 parse(ratings, {
   columns: true,
   skip_empty_lines: true
 }, async (err, records) => {
   for (const record of records) {
     await prisma.rating.create({
       data: {
         wineId: parseInt(record.wineId),
         userId: parseInt(record.userId),
         vintage: record.vintage,
         rating: parseFloat(record.rating),
         date: new Date(record.date)
       }
     })
   }
 })
}

main()
 .catch((e) => {
   console.error(e)
   process.exit(1)
 })
 .finally(async () => {
   await prisma.$disconnect()
 })
```

## 5. Database Migrations
```sql
-- prisma/migrations/YYYYMMDDHHMMSS_initial_schema/migration.sql
CREATE TABLE "Wine" (
   "id" SERIAL PRIMARY KEY,
   "name" TEXT NOT NULL,
   "type" TEXT NOT NULL,
   "elaborate" TEXT,
   "grapes" TEXT NOT NULL,
   "harmonize" TEXT NOT NULL,
   "abv" DOUBLE PRECISION NOT NULL,
   "body" TEXT NOT NULL,
   "acidity" TEXT NOT NULL,
   "code" TEXT NOT NULL,
   "price" INTEGER NOT NULL DEFAULT 0,
   "regionId" INTEGER NOT NULL,
   FOREIGN KEY ("regionId") REFERENCES "Region"("id")
);

CREATE TABLE "Region" (
   "id" SERIAL PRIMARY KEY,
   "name" TEXT NOT NULL,
   "country" TEXT NOT NULL
);

CREATE TABLE "Rating" (
   "id" SERIAL PRIMARY KEY,
   "wineId" INTEGER NOT NULL,
   "userId" INTEGER NOT NULL,
   "vintage" TEXT,
```

~~~
   "rating" DOUBLE PRECISION NOT NULL,
   "date" TIMESTAMP NOT NULL,
   FOREIGN KEY ("wineId") REFERENCES "Wine"("id"),
   FOREIGN KEY ("userId") REFERENCES "User"("id")
);

CREATE TABLE "User" (
   "id" SERIAL PRIMARY KEY,
   "email" TEXT NOT NULL UNIQUE,
   "name" TEXT,
   "password" TEXT NOT NULL,
   "role" TEXT NOT NULL DEFAULT 'USER'
);

CREATE TABLE "Image" (
   "id" SERIAL PRIMARY KEY,
   "url" TEXT NOT NULL,
   "wineId" INTEGER NOT NULL,
   FOREIGN KEY ("wineId") REFERENCES "Wine"("id")
);

-- Add indexes
CREATE INDEX idx_wine_name ON "Wine" USING gin(name gin_trgm_ops);
CREATE INDEX idx_wine_type ON "Wine"(type);
CREATE INDEX idx_wine_price ON "Wine"(price);
CREATE INDEX idx_rating_date ON "Rating"(date);
CREATE INDEX idx_rating_composite ON "Rating"(wine_id, user_id);
CREATE INDEX idx_wine_region ON "Wine"(region_id);
```

## 6. Supabase Integration
```typescript
// lib/supabase.ts
import { createClient } from '@supabase/supabase-js'

const supabaseUrl = process.env.SUPABASE_URL!
const supabaseKey = process.env.SUPABASE_ANON_KEY!

export const supabase = createClient(supabaseUrl, supabaseKey)

// RLS Policies
const policies = `
-- Allow read access to all wines
CREATE POLICY "Wines are viewable by everyone" ON "Wine"
FOR SELECT USING (true);

-- Allow insert/update for admins only
CREATE POLICY "Wines are editable by admins" ON "Wine"
FOR ALL USING (auth.role() = 'admin');

-- Users can only see their own data
CREATE POLICY "Users can only see their own data" ON "User"
FOR SELECT USING (auth.uid() = id);

-- Users can only rate once per wine
CREATE POLICY "Users can only rate once per wine" ON "Rating"
FOR INSERT WITH CHECK (
 NOT EXISTS (
   SELECT 1 FROM "Rating"
   WHERE "userId" = auth.uid()
   AND "wineId" = NEW."wineId"
 )
);
`
```

## 7. Database Operations & Error Handling
```typescript
// lib/db-operations.ts
import { PrismaClient } from '@prisma/client'

const prisma = new PrismaClient()

// Error handling wrapper
async function handleDatabaseOperation<T>(
 operation: () => Promise<T>
): Promise<T> {
 try {
   return await operation()
 } catch (error) {
   if (error.code === 'P2002') {
     throw new Error('Unique constraint violation')
   }
   if (error.code === 'P2003') {
~~~

```
      throw new Error('Foreign key constraint violation')
    }
    if (error.code === 'P2025') {
      throw new Error('Record not found')
    }
    throw error
  }
}

// Wine operations
async function getWineWithRelations(id: number) {
  return handleDatabaseOperation(() =>
    prisma.wine.findUnique({
      where: { id },
      include: {
        region: true,
        images: true,
        ratings: {
          include: {
            user: {
              select: {
                name: true
              }
            }
          }
        }
      }
    })
  )
}

// Rating operations with transaction
async function createRatingWithTransaction(
  wineId: number,
  userId: number,
  rating: number
) {
  return prisma.$transaction(async (tx) => {
    const newRating = await tx.rating.create({
      data: {
        wineId,
        userId,
        rating,
        date: new Date()
      }
    })

    const ratings = await tx.rating.findMany({
      where: { wineId }
    })

    const average = ratings.reduce((acc, r) => acc + r.rating, 0) / ratings.length

    await tx.wine.update({
      where: { id: wineId },
      data: { averageRating: average }
    })

    return newRating
  })
}
```

## 8. Batch Operations & Performance

```typescript
// lib/batch-operations.ts
// Bulk wine creation
async function bulkCreateWines(wines: WineInput[]) {
  return prisma.$transaction(
    wines.map((wine) =>
      prisma.wine.create({
        data: wine
      })
    )
  )
}

// Bulk price updates
async function bulkUpdatePrices(updates: { id: number; price: number }[]) {
  return prisma.$transaction(
    updates.map(({ id, price }) =>
      prisma.wine.update({
        where: { id },
        data: { price }
```

~~~
     })
   )
 )
}

// Optimized queries
async function getWinesWithRelations(page: number, limit: number) {
 return prisma.wine.findMany({
   take: limit,
   skip: (page - 1) * limit,
   select: {
     id: true,
     name: true,
     price: true,
     region: {
       select: {
         name: true,
         country: true
       }
     },
     images: {
       take: 1,
       select: {
         url: true
       }
     },
     _count: {
       select: {
         ratings: true
       }
     }
   }
 })
}
```

## 9. Database Maintenance
```typescript
// lib/maintenance.ts
// Health checks
async function checkDatabaseConnection() {
 try {
   await prisma.$queryRaw`SELECT 1`
   return { status: 'healthy', timestamp: new Date() }
 } catch (error) {
   return { status: 'unhealthy', error: error.message }
 }
}

// Backup
async function backupDatabase() {
 const timestamp = new Date().toISOString().replace(/[:.]/g, '-')
 const filename = `backup-${timestamp}.sql`
  const command = `pg_dump "${process.env.DIRECT_URL}" > ${filename}`
  try {
   await execAsync(command)
   return { success: true, filename }
 } catch (error) {
   return { success: false, error: error.message }
 }
}

// Cleanup
async function cleanupOldRatings(daysOld: number) {
 const cutoffDate = new Date()
 cutoffDate.setDate(cutoffDate.getDate() - daysOld)

 return prisma.rating.deleteMany({
   where: {
     date: {
       lt: cutoffDate
     }
   }
 })
}
```

## Usage Instructions

1. **Initial Setup**
```bash
npm install
cp .env.example .env
# Update .env with your credentials
~~~

~~~markdown
npx prisma generate
npx prisma migrate dev
```

2. **Seeding Data**
```bash
npm run seed
```

3. **Database Operations**
```bash
# Start Prisma Studio
npx prisma studio

# Create new migration
npx prisma migrate dev --name describe_your_change

# Push to production
npx prisma migrate deploy
```

4. **Backup and Maintenance**
```bash
# Backup database
node scripts/backup.js

# Check database health
node scripts/health-check.js

# Clean old data
node scripts/cleanup.js
```

## Common Issues and Solutions

1. **Connection Issues**
- Check DATABASE_URL and DIRECT_URL in .env
- Verify Supabase credentials
- Ensure proper network access

2. **Migration Issues**
```bash
# Reset database (development only)
npx prisma migrate reset

# Resolve migration history
npx prisma migrate resolve
```

3. **Performance Issues**
- Use included indexes
- Monitor query statistics
- Implement caching as needed

## Security Considerations

1. Always use environment variables for sensitive data
2. Implement proper RLS policies in Supabase
3. Use transactions for critical operations
4. Regular backup maintenance
5. Monitor database performance
~~~