

# Building a Wine Store Web Application: A Complete Tutorial

## Version 1.0

*A comprehensive guide to building a full-stack wine store application using modern web development tools*

## Table of Contents

1. Prerequisites and Setup
2. Understanding Modern Web Development
3. Project Setup and Initial Configuration
4. Database Design and Setup
5. Building the Backend
6. Creating the Frontend
7. Styling and UI
8. Deployment
9. Next Steps and Advanced Topics

## 1. Prerequisites and Setup

Before we begin building our wine store application, we need to set up our development environment. Let's install all the necessary software and tools.

First, download and install Node.js from [nodejs.org](https://nodejs.org). Choose the LTS (Long Term Support) version. After installation, open your terminal and verify the installation by running:

```
node --version npm --version
```

You should see version numbers displayed for both commands.

Next, install Visual Studio Code from [code.visualstudio.com](https://code.visualstudio.com). After installation, open VS Code and install these essential extensions:

- ESLint for code linting
- Prettier for code formatting
- Prisma for database management
- TypeScript for type support
- GitLens for better Git integration

For our database, we'll use PostgreSQL. Download it from [postgres.org](https://postgres.org) and during installation, make sure to:

- Note down your superuser password
- Keep the default port (5432)
- Install all offered components

After installation, verify PostgreSQL is working by running: `psql --version`

Finally, install Git from [git-scm.com](https://git-scm.com). After installation, configure your Git identity by running:

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

## 2. Understanding Modern Web Development

Our wine store will use several modern web technologies working together. Let's understand each one:

**JavaScript and TypeScript:** JavaScript is the language of the web, while TypeScript adds type safety. Here's a simple example showing the difference:

```
In JavaScript: function calculateTotal(price, quantity) { return price * quantity; }

In TypeScript: function calculateTotal(price: number, quantity: number): number { return price * quantity; }

React is our main UI library. It lets us build interfaces using components. A simple React component looks like this:

function WineCard({ name, price, description }) { return (

  {name}

  {description}

  ${price}
); }
```

Next.js builds on top of React and provides:

- Automatic page routing
- Server-side rendering
- API routes
- Image optimization
- Development server

Our database will use PostgreSQL with Prisma as our ORM (Object-Relational Mapping). This combination allows us to:

- Store wine data securely
- Create relationships between data
- Query data efficiently
- Maintain data integrity

### 3. Project Setup

---

Let's create our project. Open your terminal and run:

```
npx create-next-app@latest store-wine
```

When prompted, answer: Would you like to use TypeScript? Yes Would you like to use ESLint? Yes Would you like to use Tailwind CSS? Yes Would you like to use src/ directory? No Would you like to use App Router? Yes Would you like to customize the default import alias? No

This creates our project with this structure: store-wine/ ├── app/ | └─ page.tsx ├── public/ ├── styles/ ├── .gitignore ├── next.config.js ├── package.json ├── README.md └─ tsconfig.json

Now let's set up our database connection. Install Prisma by running:

```
npm install prisma --save-dev npm install @prisma/client
```

Initialize Prisma in your project:

```
npx prisma init
```

This creates a prisma directory with a schema.prisma file and a .env file. Open .env and add your database connection:

```
DATABASE_URL="postgresql://username:password@localhost:5432/wine_store"
```

[Continuing in next message...]

### 4. Database Setup

---

Now we'll create our database schema. Open prisma/schema.prisma and replace its contents with our complete wine store schema:

```
generator client { provider = "prisma-client-js" output = "../lib/generated/prisma" }

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 @unique price Int @default(0) inStock Boolean
@default(true) stockCount Int @default(0) createdAt DateTime @default(now()) updatedAt DateTime @updatedAt
region Region @relation(fields: [regionId], references: [id]) regionId Int ratings Rating[] images Image[] }
```

Add the remaining models for Region, Rating, User, and Image as shown earlier.

Next, set up your Supabase database. Go to [supabase.com](https://supabase.com) and:

1. Create a new account
2. Create a new project
3. Note down your database credentials
4. Update your .env file with:

```
DATABASE_URL="postgres://postgres:[YOUR-PASSWORD]@db.[YOUR-PROJECT-REF].supabase.co:5432/postgres" DIRECT_URL="postgres://postgres:[YOUR-PASSWORD]@db.[YOUR-PROJECT-REF].supabase.co:5432/postgres" NEXT_PUBLIC_SUPABASE_URL="https://[YOUR-PROJECT-REF].supabase.co" NEXT_PUBLIC_SUPABASE_ANON_KEY="your-anon-key"
```

Initialize your database by running: `npx prisma db push` `npx prisma generate`

## 5. Building the Backend

Create your API routes structure like this:

```
app/ └─ api/ └─ auth/ | └─ login/ | | └─ route.ts | └─ register/ | └─ route.ts └─ wines/ | └─ [id]/ | | └─ route.ts | └─ route.ts └─ ratings/ └─ route.ts
```

Install authentication packages: `npm install @supabase/supabase-js bcryptjs` `npm install @types/bcryptjs --save-dev`

Create `utils/auth.ts` for authentication:

```
import { createClient } from '@supabase/supabase-js'; import bcrypt from 'bcryptjs';

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

export const supabase = createClient(supabaseUrl, supabaseKey);

export async function hashPassword(password: string): Promise { return bcrypt.hash(password, 10); }

export async function verifyPassword(password: string, hashedPassword: string): Promise { return bcrypt.compare(password, hashedPassword); }
```

Create your wine listing API in `app/api/wines/route.ts`:

```
import { NextResponse } from 'next/server'; import { prisma } from '@lib/prisma';

export async function GET(request: Request) { try { const wines = await prisma.wine.findMany({ include: { region: true, images: true, ratings: { include: { user: { select: { name: true, email: true } } } } }); return NextResponse.json(wines); } catch (error) { return NextResponse.json({ error: 'Failed to fetch wines' }, { status: 500 }); } }

export async function POST(request: Request) { try { const data = await request.json(); const wine = await prisma.wine.create({ data, include: { region: true, images: true } }); return NextResponse.json(wine); } catch (error) { return NextResponse.json({ error: 'Failed to create wine' }, { status: 500 }); } }
```

[Continuing in next message...]

## 6. Creating the Frontend

---

Set up your component structure as follows:


```
components/ | — layout/ | — Header.tsx | — Footer.tsx | — Navigation.tsx | — wine/ | — WineCard.tsx |
| — WineList.tsx | — WineDetail.tsx | — auth/ | — LoginForm.tsx | — RegisterForm.tsx | — ui/ | —
Button.tsx | — Input.tsx | — Card.tsx
```

Create your WineCard component in components/wine/WineCard.tsx:

```
import Image from 'next/image'; import { Wine } from '@prisma/client';

interface WineCardProps { wine: Wine; onSelect: (wine: Wine) => void; }

export function WineCard({ wine, onSelect }: WineCardProps) { return (

onSelect(wine))> {wine.images?.[0] && ( {wine.name} )}

{wine.name}

{wine.type}

${wine.price.toFixed(2)}

); }
```

Create your WineList component in components/wine/WineList.tsx:

```
import { useEffect, useState } from 'react'; import { Wine } from '@prisma/client'; import { WineCard } from
'./WineCard';

export function WineList() { const [wines, setWines] = useState<Wine[]>([]); const [loading, setLoading] =
useState(true);

useEffect(() => { async function fetchWines() { try { const response = await fetch('/api/wines'); const data = await
response.json(); setWines(data); } catch (error) { console.error('Failed to fetch wines:', error); } finally {
setLoading(false); } } fetchWines(); }, []);

if (loading) return

Loading...

;

return (

{wines.map((wine) => ( { // Handle wine selection } } /> ))}

); }
```

## 7. Styling and UI

---

Install Tailwind CSS plugins: `npm install -D @tailwindcss/forms @tailwindcss/typography`

Update your tailwind.config.js:

```
module.exports = { content: [ './pages/**/*.{js,ts,jsx,tsx,mdx}', './components/**/*.{js,ts,jsx,tsx,mdx}', './app/**/*.
{js,ts,jsx,tsx,mdx}', ], theme: { extend: { colors: { primary: '#7C3AED', secondary: '#5B21B6', }, }, }, plugins: [
require('@tailwindcss/forms'), require('@tailwindcss/typography'), ], }
```

Create app/globals.css:

```
@tailwind base; @tailwind components; @tailwind utilities;

@layer components { .btn-primary { @apply bg-primary text-white px-4 py-2 rounded-md hover:bg-secondary
transition-colors; }
```

`.input-field { @apply border-gray-300 rounded-md shadow-sm focus:ring-primary focus:border-primary; }`

## 8. Deployment

---

Update next.config.js for production:

```
const nextConfig = { images: { domains: ['your-supabase-storage-domain.supabase.co'], }, experimental: { serverActions: true, }, }
```

```
module.exports = nextConfig
```

Build your application: `npm run build`

Install Vercel CLI: `npm install -g vercel`

Deploy to Vercel: `vercel`

Add these environment variables in your Vercel dashboard:

- DATABASE\_URL
- DIRECT\_URL
- NEXT\_PUBLIC\_SUPABASE\_URL
- NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY

## 9. Next Steps and Advanced Topics

---

Consider implementing these features next:

1. Shopping Cart:

- Add to cart functionality
- Cart persistence
- Checkout process

2. User Features:

- User reviews and ratings
- Favorite wines
- Purchase history

3. Admin Dashboard:

- Inventory management
- Order processing
- User management

4. Performance Improvements:

- Implement caching
- Add pagination
- Optimize images
- Use React Suspense
- Add infinite scrolling

5. Testing Setup: Install testing packages: `npm install -D jest @testing-library/react @testing-library/jest-dom`

```
Create jest.config.js: module.exports = { testEnvironment: 'jsdom', setupFilesAfterEnv: ['./jest.setup.js'], testPathIgnorePatterns: ['/.next/', '/node_modules/'], };
```

## Resources and References

---

### Official Documentation:

- Next.js: [nextjs.org/docs](https://nextjs.org/docs)
- Prisma: [prisma.io/docs](https://prisma.io/docs)
- Supabase: [supabase.com/docs](https://supabase.com/docs)
- Tailwind CSS: [tailwindcss.com/docs](https://tailwindcss.com/docs)

### Learning Resources:

- React: [react.dev](https://react.dev)
- TypeScript: [typescriptlang.org/docs](https://typescriptlang.org/docs)
- PostgreSQL: [postgresql.org/docs/current/tutorial.html](https://postgresql.org/docs/current/tutorial.html)

### Community Resources:

- Next.js GitHub: [github.com/vercel/next.js](https://github.com/vercel/next.js)
- Prisma GitHub: [github.com/prisma/prisma](https://github.com/prisma/prisma)
- Supabase GitHub: [github.com/supabase/supabase](https://github.com/supabase/supabase)

*Note: This tutorial is Version 1.0 and will be updated with more content and improvements in future versions.*