🛒 Full-Stack E-Commerce App

A complete e-commerce application built with **Next.js**, **TypeScript**, **Prisma**, **Stripe**, and **Tailwind CSS**.

✨ Features

🛍️ Customer Features

- Browse products with search and category filtering
- Add products to cart with quantity management
- Secure checkout with Stripe payment processing
- User authentication (register/login)
- Order history and tracking

👨‍💼 Admin Features

- Complete product management (CRUD operations)
- Order management with status updates
- Real-time inventory tracking
- Sales analytics dashboard
- Role-based access control

💳 Payment Integration

- Stripe Checkout for secure payments
- Webhook handling for order automation
- Automatic stock updates after purchase
- Test mode ready with demo cards

🔐 Security

- JWT-based authentication
- HTTP-only cookies
- Password hashing with bcrypt
- Protected API routes
- Role-based authorization

🏗️ Tech Stack

- **Frontend**: React, Next.js 14, TypeScript, Tailwind CSS
- **Backend**: Next.js API Routes, Prisma ORM
- **Database**: PostgreSQL
- **Authentication**: JWT, bcryptjs
- **Payments**: Stripe
- **Icons**: Lucide React

📁 Project Structure

```
├── pages/
│   ├── api/
│   │   ├── products/
│   │   │   ├── index.ts        # GET, POST products
│   │   │   └── [id].ts         # GET, PUT, DELETE product
│   │   ├── orders/
│   │   │   ├── index.ts        # GET, POST orders
│   │   │   └── [id].ts         # GET, PUT order
│   │   ├── auth/
│   │   │   ├── register.ts     # User registration
│   │   │   ├── login.ts        # User login
│   │   │   ├── logout.ts       # User logout
│   │   │   └── me.ts           # Get current user
│   │   ├── checkout_sessions.ts  # Create Stripe checkout
│   │   └── webhooks/
│   │       └── stripe.ts       # Stripe webhook handler
│   ├── checkout/
│   │   ├── success.tsx         # Payment success page
│   │   └── cancel.tsx          # Payment cancel page
│   └── index.tsx               # Main app component
├── prisma/
│   ├── schema.prisma           # Database schema
│   └── seed.ts                 # Database seeding
├── .env.example                # Environment variables template
├── package.json
└── README.md
```

🚀 Quick Start

Prerequisites

- Node.js 18+ and npm
- PostgreSQL database
- Stripe account (test mode)

Installation

1. **Clone and install dependencies**

```bash
npm install
```

2. **Set up environment variables**

```bash
cp .env.example .env
```

Update `.env` with your values:

```env
DATABASE_URL="postgresql://user:password@localhost:5432/ecommerce"
STRIPE_PUBLISHABLE_KEY="pk_test_..."
STRIPE_SECRET_KEY="sk_test_..."
STRIPE_WEBHOOK_SECRET="whsec_..."
JWT_SECRET="your-secret-key"
```

Note: Test Stripe keys are already configured in the code for demo purposes.

3. **Initialize database**

```bash
# Run migrations
npx prisma migrate dev --name init

# Generate Prisma Client
npx prisma generate

# Seed database with sample data
npx prisma db seed
```

4. **Set up Stripe webhooks (for local testing)**

```bash
# Install Stripe CLI
brew install stripe/stripe-cli/stripe  # macOS
# or download from https://stripe.com/docs/stripe-cli

# Login to Stripe
stripe login

# Forward webhooks to local server
stripe listen --forward-to localhost:3000/api/webhooks/stripe
```

Copy the webhook signing secret (whsec_...) to your `.env` file.

5. **Run development server**

```bash
npm run dev
```

Visit http://localhost:3000

🧪 Testing

Test Credentials

After seeding, you can use:

Admin Account

- Email: `admin@example.com`
- Password: `admin123`

Customer Account

- Email: `customer@example.com`
- Password: `customer123`

Test Payment Cards

Use these Stripe test cards in checkout:

Card Number	Result
4242 4242 4242 4242	Success
4000 0000 0000 0002	Card declined
4000 0000 0000 9995	Insufficient funds

Use any future expiry date and any 3-digit CVC.

Testing Workflow

1. **Register/Login** as a customer
2. **Browse products** and add items to cart
3. **Checkout** - you'll be redirected to Stripe
4. **Enter test card**: `4242 4242 4242 4242`
5. **Complete payment** - webhook automatically creates order
6. **Login as admin** to view and manage orders
7. **Update order status** to track fulfillment

📡 API Documentation

Products

```typescript
GET    /api/products          // Get all products
POST   /api/products          // Create product (admin)
GET    /api/products/:id      // Get single product
PUT    /api/products/:id      // Update product (admin)
DELETE /api/products/:id      // Delete product (admin)
```

Orders

```typescript
GET    /api/orders            // Get all orders (admin)
POST   /api/orders            // Create order (webhook)
GET    /api/orders/:id        // Get single order
PUT    /api/orders/:id        // Update order status (admin)
```

Authentication

```typescript
```

```
POST   /api/auth/register        // Register new user
POST   /api/auth/login           // Login user
POST   /api/auth/logout          // Logout user
GET    /api/auth/me              // Get current user
```

Payment

typescript

```
POST   /api/checkout_sessions     // Create Stripe checkout
POST   /api/webhooks/stripe       // Stripe webhook handler
```

🗄️ Database Schema

prisma

```
model User {
  id       String   @id @default(cuid())
  email    String   @unique
  name     String?
  password String
  role     Role     @default(CUSTOMER)
  orders   Order[]
}

model Product {
  id          Int      @id @default(autoincrement())
  name        String   @unique
  price       Float
  stock       Int
  category    String
  image       String
  description String?
}

model Order {
  id         Int         @id @default(autoincrement())
  customer   String
  email      String
  total      Float
  status     OrderStatus
  orderItems OrderItem[]
}

model OrderItem {
  id        Int     @id @default(autoincrement())
  orderId   Int
  productId Int
  quantity  Int
  price     Float
}
```

🔧 Configuration

Stripe Setup

1. Get API keys from Stripe Dashboard
2. Add keys to `.env` file

3. For local testing, use Stripe CLI to forward webhooks
4. For production, configure webhook endpoint in Stripe Dashboard

Database Setup

Supports PostgreSQL, MySQL, SQLite, and SQL Server. Update `provider` in `schema.prisma`:

```prisma
datasource db {
  provider = "postgresql" // or "mysql", "sqlite", "sqlserver"
  url      = env("DATABASE_URL")
}
```

🚢 Production Deployment

Deploy to Vercel

1. Push code to GitHub
2. Import project in Vercel
3. Add environment variables
4. Deploy

Production Checklist

- [ ] Set production DATABASE_URL
- [ ] Use Stripe live API keys
- [ ] Configure production webhook URL in Stripe
- [ ] Set strong JWT_SECRET
- [ ] Enable HTTPS
- [ ] Add rate limiting
- [ ] Set up monitoring and logging
- [ ] Configure email notifications
- [ ] Add error tracking (e.g., Sentry)
- [ ] Implement backup strategy

🤝 Contributing

Contributions are welcome! Please feel free to submit a Pull Request.

📝 License

MIT License - feel free to use this project for learning or commercial purposes.

🙏 Acknowledgments

- Next.js team for the amazing framework
- Stripe for payment processing
- Prisma for the excellent ORM
- Tailwind CSS for styling
- Lucide for beautiful icons

📧 Support

For questions or issues, please open an issue on GitHub.

Built with ❤️ using Next.js, TypeScript, Prisma, and Stripe