E-Commerce App Setup Instructions

🚀 Complete Setup Guide

1. Install Dependencies

bash

```
npm install
```

2. Database Setup

bash

```
# Create .env file (copy from .env.example)
cp .env.example .env

# Update DATABASE_URL in .env with your PostgreSQL connection string
# Example: postgresql://user:password@localhost:5432/ecommerce

# Run Prisma migrations
npx prisma migrate dev --name init

# Generate Prisma Client
npx prisma generate
```

3. Seed Database (Optional)

Create a `prisma/seed.ts` file to add initial products:

typescript

```
import { PrismaClient } from '@prisma/client';
import bcrypt from 'bcryptjs';

const prisma = new PrismaClient();

async function main() {
  // Create admin user
  const adminPassword = await bcrypt.hash('admin123', 10);
  const admin = await prisma.user.create({
    data: {
      email: 'admin@example.com',
      password: adminPassword,
      name: 'Admin User',
      role: 'ADMIN',
    },
  });

  // Create sample products
  const products = [
    { name: 'Wireless Headphones', price: 79.99, image: '🎧', stock: 15, category: 'Electronics' },
    { name: 'Smart Watch', price: 199.99, image: '⌚', stock: 8, category: 'Electronics' },
    { name: 'Laptop Sleeve', price: 29.99, image: '💼', stock: 25, category: 'Accessories' },
    { name: 'USB-C Cable', price: 12.99, image: '🔌', stock: 50, category: 'Accessories' },
    { name: 'Bluetooth Speaker', price: 49.99, image: '🔊', stock: 12, category: 'Electronics' },
    { name: 'Phone Case', price: 19.99, image: '📱', stock: 30, category: 'Accessories' },
  ];

  for (const product of products) {
    await prisma.product.create({ data: product });
  }
```

```
  console.log('Database seeded successfully!');
}

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

Add to `package.json`:

json

```
"prisma": {
  "seed": "ts-node --compiler-options {\"module\":\"CommonJS\"} prisma/seed.ts"
}
```

Run seed:

bash

```
npx prisma db seed
```

4. Stripe Setup

1. **Get your API keys** from Stripe Dashboard

- Already configured in the code with test keys
- Publishable Key: `pk_test_51R8wQ5Qp1gA9BIqH5EJXpFQBxRhpohWcvpwjlm3kQbgIJjq0fBiHvJMPmNA94wI3UzLvEkoKsKMgAlmz9eYWUUy500396leQnv`
- Secret Key: `sk_test_51R8wQ5Qp1gA9BIqHAJG1DPNbRG9GwyGd5JBTnbsDfXgCSk9SLAQvy38GvNhLlqAd5dlJxL1BM0ShjAIxbcMuBett00A2IWQ00l`

2. **Test Webhook Locally** using Stripe CLI:

bash

```
# Install Stripe CLI
# macOS: brew install stripe/stripe-cli/stripe
# Windows: scoop install stripe
# 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
```

3. **Test Cards** (use these in checkout):
   - Success: `4242 4242 4242 4242`
   - Decline: `4000 0000 0000 0002`
   - Any future expiry date and any 3-digit CVC

5. Run Development Server

bash

```
npm run dev
```

Visit http://localhost:3000

📋 Features Implemented

✅ Product Management

- CRUD operations via API
- Search and category filtering
- Real-time stock updates
- Admin panel for management

✅ Orders System

- View all orders in admin
- Update order status
- Track order history
- Customer information

✅ Stripe Integration

- Checkout session creation
- Secure payment processing
- Webhook handling for order creation
- Automatic stock updates after payment

✅ Authentication

- User registration and login
- JWT-based authentication
- HTTP-only cookies for security
- Role-based access (Admin/Customer)
- Protected admin routes

🔐 Default Admin Credentials

After seeding:

- Email: `admin@example.com`

- Password: `admin123`

🧪 Testing Workflow

1. **Register a new user** or login with admin credentials
2. **Browse products** and add to cart
3. **Proceed to checkout** - redirects to Stripe
4. **Use test card**: `4242 4242 4242 4242`
5. **Complete payment** - webhook creates order automatically
6. **Check admin panel** - order appears with status "PROCESSING"
7. **Update order status** to "COMPLETED"

📡 API Endpoints

Products

- `GET /api/products` - Fetch all products
- `POST /api/products` - Create product
- `GET /api/products/[id]` - Get single product
- `PUT /api/products/[id]` - Update product
- `DELETE /api/products/[id]` - Delete product

Orders

- `GET /api/orders` - Fetch all orders
- `POST /api/orders` - Create order (webhook)
- `GET /api/orders/[id]` - Get single order
- `PUT /api/orders/[id]` - Update order status

Authentication

- `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

- `POST /api/checkout_sessions` - Create Stripe checkout

- `POST /api/webhooks/stripe` - Stripe webhook handler

🛠️ Troubleshooting

Database Connection Issues

- Ensure PostgreSQL is running
- Check DATABASE_URL in .env
- Run `npx prisma studio` to verify database

Stripe Webhook Not Working

- Make sure Stripe CLI is running
- Check webhook secret in .env matches CLI output
- Verify endpoint URL is correct

Authentication Not Working

- Clear browser cookies
- Check JWT_SECRET is set in .env
- Verify token is being sent in requests

🚢 Production Deployment

1. **Environment Variables**: Set all production values
2. **Database**: Use production PostgreSQL instance
3. **Stripe**: Switch to live API keys
4. **Webhook**: Configure production webhook URL in Stripe Dashboard
5. **Security**:
    - Use strong JWT_SECRET
    - Enable HTTPS
    - Set secure cookie flags
    - Add rate limiting

📝 Notes

- Test keys are already in the code (safe for demo)
- In production, move all keys to environment variables

- Implement proper error boundaries
- Add input validation on frontend
- Consider adding email notifications
- Implement inventory management
- Add order tracking for customers