E-Commerce Test Application

Comprehensive Guide for Page Object Model Testing

📁 Project Structure

```
TestingWebApplication/
├── 📁 frontend/                    # React TypeScript frontend
│   ├── src/
│   │   ├── components/             # Reusable UI components
│   │   │   ├── LoadingSpinner.tsx
│   │   │   ├── Navbar.tsx
│   │   │   └── ProtectedRoute.tsx
│   │   ├── contexts/               # React context providers
│   │   │   └── AuthContext.tsx
│   │   ├── pages/                  # Page components
│   │   │   ├── HomePage.tsx
│   │   │   ├── LoginPage.tsx
│   │   │   ├── RegisterPage.tsx
│   │   │   ├── ProductsPage.tsx
│   │   │   ├── ProductDetailPage.tsx
│   │   │   ├── CartPage.tsx
│   │   │   ├── CheckoutPage.tsx
│   │   │   ├── ProfilePage.tsx
│   │   │   ├── OrdersPage.tsx
│   │   │   └── AdminDashboard.tsx
│   │   ├── App.tsx                 # Main app component
│   │   ├── index.tsx               # React entry point
│   │   └── index.css               # Tailwind CSS styles
│   ├── package.json                # Frontend dependencies
│   ├── tsconfig.json               # TypeScript configuration
│   ├── tailwind.config.js          # Tailwind CSS configuration
│   └── Dockerfile                  # Frontend container configuration
├── 📁 backend/                     # Node.js Express API
│   ├── src/
│   │   ├── middleware/             # Express middleware
│   │   │   └── auth.ts             # JWT authentication middleware
│   │   ├── routes/                 # API route handlers
│   │   │   ├── auth.ts             # Authentication endpoints
│   │   │   ├── products.ts         # Product CRUD operations
│   │   │   ├── categories.ts       # Category endpoints
│   │   │   ├── cart.ts             # Shopping cart operations
│   │   │   ├── orders.ts           # Order management
│   │   │   └── users.ts            # User profile endpoints
│   │   ├── index.ts                # Express server setup
│   │   └── seed.ts                 # Database seeding script
│   ├── prisma/
│   │   └── schema.prisma           # Database schema definition
│   ├── package.json                # Backend dependencies
```

```
│   ├── tsconfig.json              # TypeScript configuration
│   ├── .env.example               # Environment variables template
│   └── Dockerfile                 # Backend container configuration
├── 📁 database/                    # Database configuration
│   └── init.sql                   # PostgreSQL initialization script
├── 📁 .github/                     # GitHub configuration
│   └── copilot-instructions.md    # AI coding assistant instructions
├── 📁 .vscode/                     # VS Code configuration
│   └── tasks.json                 # Development tasks
├── 🐳 docker-compose.yml           # Container orchestration
├── 📖 README.md                   # Project documentation
└── .gitignore                     # Git ignore rules
```

🚀 Key Features for Testing

Authentication System

- ✅ **User Registration**: Complete form with validation
  - Email validation
  - Password strength requirements
  - Duplicate email handling
  - Success/error feedback
- ✅ **User Login**: Authentication with error handling
  - Email and password validation
  - Invalid credential handling
  - JWT token management
  - Automatic logout on token expiration
- ✅ **Role-Based Access Control**
  - USER role: Standard e-commerce functionality
  - ADMIN role: Administrative dashboard access
  - Protected route redirects
  - Permission-based UI elements

E-Commerce Features

- ✅ **Product Catalog**
  - Product listing with pagination
  - Search functionality
  - Category filtering
  - Price range filtering
  - Product detail views
- ✅ **Shopping Cart**

    - Add products to cart
    - Update item quantities
    - Remove items from cart
    - Cart persistence across sessions
    - Real-time cart total calculations

- ✅ **Checkout Process**
    - Order summary review
    - Order placement
    - Order confirmation
    - Order history tracking

- ✅ **User Profile Management**
    - View profile information
    - Update account details
    - Order history access

- ✅ **Admin Dashboard**
    - Product management (CRUD operations)
    - User management
    - Order monitoring
    - Category management

Testing-Friendly Design

- ✅ **Comprehensive data-testid Attributes**
    - All interactive elements tagged
    - Consistent naming conventions
    - Form fields and buttons identified
    - Navigation elements tagged

- ✅ **Error States and Loading States**
    - Loading spinners for async operations
    - Error message displays
    - Form validation feedback
    - Network error handling

- ✅ **Clear Page Navigation Structure**
    - Predictable URL patterns
    - Breadcrumb navigation
    - Protected route handling
    - Role-based menu items

🛠️ Technology Stack

Frontend Technologies

- **React 18**: Latest React version with modern hooks
- **TypeScript**: Type safety and better development experience
- **Tailwind CSS**: Utility-first CSS framework for styling
- **React Router**: Client-side routing and navigation
- **React Query**: Server state management and caching
- **Axios**: HTTP client for API requests
- **React Hot Toast**: User-friendly notification system

Backend Technologies

- **Node.js**: JavaScript runtime environment
- **Express**: Web application framework
- **TypeScript**: Type-safe server-side development
- **Prisma ORM**: Database toolkit and query builder
- **PostgreSQL**: Robust relational database
- **JWT**: JSON Web Tokens for authentication
- **bcryptjs**: Password hashing and validation
- **Joi**: Request validation library

Infrastructure

- **Docker Compose**: Multi-container application orchestration
- **PostgreSQL Database**: Persistent data storage
- **Nginx**: Production-ready web server (configurable)
- **VS Code**: Integrated development environment support

🎯 Perfect Test Scenarios

1. User Registration & Authentication Flow

```
Test Case: Complete User Registration
├── Navigate to registration page
├── Fill out registration form
│   ├── Enter valid email address
│   ├── Enter secure password
│   ├── Enter first and last name
│   └── Submit form
├── Verify successful registration
├── Check automatic login
└── Verify user dashboard access

Test Case: Login with Valid Credentials
├── Navigate to login page
├── Enter registered email and password
├── Submit login form
├── Verify successful authentication
├── Check JWT token storage
```

```
└── Verify protected route access

Test Case: Login with Invalid Credentials
├── Navigate to login page
├── Enter invalid email or password
├── Submit login form
├── Verify error message display
├── Check form remains populated
└── Verify no authentication occurs
```

2. Product Browsing & Search

```
Test Case: Browse Product Catalog
├── Navigate to products page
├── Verify product grid display
├── Check product card information
├── Test pagination functionality
└── Verify product detail navigation

Test Case: Search Products
├── Enter search term in search box
├── Submit search query
├── Verify filtered results
├── Test empty search results
└── Clear search and verify reset

Test Case: Filter Products by Category
├── Select category filter
├── Verify filtered product display
├── Test multiple filter combinations
├── Clear filters and verify reset
└── Test price range filtering
```

3. Shopping Cart Management

```
Test Case: Add Products to Cart
├── Navigate to product detail page
├── Select product options (if any)
├── Click "Add to Cart" button
├── Verify cart icon update
├── Navigate to cart page
└── Verify product in cart

Test Case: Update Cart Quantities
├── Navigate to cart page
├── Modify item quantity
├── Verify price recalculation
├── Test quantity limits
└── Verify cart total updates
```

```
Test Case: Remove Items from Cart
├── Navigate to cart page
├── Click remove item button
├── Verify item removal
├── Check cart total update
└── Test empty cart state
```

4. Checkout Process

```
Test Case: Complete Purchase Flow
├── Add items to cart
├── Navigate to checkout
├── Review order summary
├── Confirm order placement
├── Verify order confirmation
├── Check cart clearance
└── Verify order in history

Test Case: Checkout with Empty Cart
├── Navigate to checkout with empty cart
├── Verify appropriate error handling
└── Redirect to products page
```

5. Admin Functions

```
Test Case: Admin Dashboard Access
├── Login with admin credentials
├── Verify admin menu visibility
├── Navigate to admin dashboard
├── Check admin-only features
└── Verify user management access

Test Case: Product Management
├── Access admin product management
├── Create new product
├── Edit existing product
├── Delete product
└── Verify changes in catalog

Test Case: Role-Based Access Control
├── Login as regular user
├── Attempt admin dashboard access
├── Verify access denial
├── Check hidden admin menu items
└── Test protected admin routes
```

🧪 Test Data & Accounts

Pre-configured Test Accounts

```
Admin Account:
├── Email: admin@ecommerce.com
├── Password: admin123
├── Role: ADMIN
└── Permissions: Full system access

Standard User Account:
├── Email: user@test.com
├── Password: user123
├── Role: USER
└── Permissions: Standard e-commerce features
```

Sample Product Data

The application includes comprehensive sample data:

Categories:

- Electronics (MacBook Pro, iPhone, Headphones, iPad)
- Clothing (T-Shirts, Denim Jackets, Running Shoes)
- Books (Programming, Design, Technical manuals)

Product Attributes:

- Unique product IDs
- Names and descriptions
- Price points ($29.99 - $2,499.00)
- Stock quantities
- High-quality placeholder images
- Featured product flags
- Category associations

📊 API Endpoints Reference

Authentication Endpoints

```
POST /api/auth/register
├── Body: { email, password, firstName, lastName }
├── Returns: { user, token }
└── Status: 201 Created | 400 Bad Request | 409 Conflict

POST /api/auth/login
├── Body: { email, password }
```

```
  ├── Returns: { user, token }
  └── Status: 200 OK | 401 Unauthorized
```

Product Endpoints

```
GET /api/products
├── Query: ?category&search&minPrice&maxPrice&page&limit
├── Returns: { products[], pagination }
└── Status: 200 OK

GET /api/products/:id
├── Returns: { product }
└── Status: 200 OK | 404 Not Found

POST /api/products (Admin Only)
├── Body: { name, description, price, categoryId, ... }
├── Returns: { product }
└── Status: 201 Created | 403 Forbidden

PUT /api/products/:id (Admin Only)
├── Body: { name, description, price, ... }
├── Returns: { product }
└── Status: 200 OK | 403 Forbidden | 404 Not Found

DELETE /api/products/:id (Admin Only)
├── Returns: { message }
└── Status: 200 OK | 403 Forbidden | 404 Not Found
```

Cart Management Endpoints

```
GET /api/cart
├── Headers: Authorization: Bearer <token>
├── Returns: { cartItems[] }
└── Status: 200 OK | 401 Unauthorized

POST /api/cart/add
├── Headers: Authorization: Bearer <token>
├── Body: { productId, quantity }
├── Returns: { cartItem }
└── Status: 201 Created | 401 Unauthorized

PUT /api/cart/:id
├── Headers: Authorization: Bearer <token>
├── Body: { quantity }
├── Returns: { cartItem }
└── Status: 200 OK | 401 Unauthorized

DELETE /api/cart/:id
├── Headers: Authorization: Bearer <token>
```

```
  ├── Returns: { message }
  └── Status: 200 OK | 401 Unauthorized
```

Order Management Endpoints

```
GET /api/orders
├── Headers: Authorization: Bearer <token>
├── Returns: { orders[] }
└── Status: 200 OK | 401 Unauthorized

POST /api/orders/checkout
├── Headers: Authorization: Bearer <token>
├── Returns: { order }
└── Status: 201 Created | 400 Bad Request | 401 Unauthorized
```

🚢 Getting Started

Prerequisites

Before running the application, ensure you have one of the following setups:

Option 1: Docker Setup (Recommended)

- Docker Desktop installed and running
- Docker Compose available
- Minimum 4GB RAM allocated to Docker
- Ports 3000, 5000, and 5432 available

Option 2: Manual Setup

- Node.js 18+ installed
- npm or yarn package manager
- PostgreSQL 12+ database server
- Git for version control

Quick Start with Docker

1. **Clone and Navigate**

   ```
   git clone <repository-url>
   cd TestingWebApplication
   ```

2. **Start All Services**

   ```
   docker-compose up --build
   ```

3. **Wait for Services to Start**
   - Database initialization: ~30 seconds
   - Backend API server: ~60 seconds
   - Frontend development server: ~90 seconds
4. **Access the Application**
   - **Frontend**: http://localhost:3000
   - **Backend API**: http://localhost:5000
   - **Database**: localhost:5432
5. **Verify Setup**
   - Navigate to http://localhost:3000
   - Click "Login" and use test credentials
   - Explore the application features

Manual Development Setup

1. **Database Setup**

```
# Install and start PostgreSQL
# Create database: ecommerce_db
# Create user: ecommerce_user
```

2. **Backend Setup**

```
cd backend
cp .env.example .env
# Edit .env with your database credentials
npm install
npx prisma migrate dev
npx prisma generate
npm run seed
npm run dev
```

3. **Frontend Setup**

```
cd frontend
npm install
npm start
```

Environment Configuration

Backend Environment Variables (.env)

```
DATABASE_URL=postgresql://ecommerce_user:ecommerce_password@localhost:5432
/ecommerce_db
JWT_SECRET=your-super-secret-jwt-key-change-in-production
NODE_ENV=development
PORT=5000
FRONTEND_URL=http://localhost:3000
```

Frontend Environment Variables (.env)

```
REACT_APP_API_URL=http://localhost:5000/api
```

VS Code Integration

The project includes VS Code tasks for streamlined development:

Available Tasks:

- "Build and Run E-Commerce App": Starts Docker Compose
- Backend development server
- Frontend development server
- Database migrations
- Test execution

Usage:

1. Open VS Code in project directory
2. Press `Cmd/Ctrl + Shift + P`
3. Type "Tasks: Run Task"
4. Select desired task from the list

Troubleshooting Common Issues

Port Conflicts:

- Check if ports 3000, 5000, or 5432 are in use
- Stop conflicting services or change ports in docker-compose.yml

Docker Build Failures:

- Ensure Docker Desktop is running
- Clear Docker cache: `docker system prune -a`
- Rebuild containers: `docker-compose up --build --force-recreate`

Database Connection Issues:

- Verify PostgreSQL container is running

- Check database credentials in environment variables
- Wait for database initialization to complete

Frontend/Backend Communication:

- Verify backend is running on port 5000
- Check CORS configuration in backend
- Confirm API URL in frontend environment variables

📝 Additional Resources

Development Tools

- **VS Code Extensions**: TypeScript, Tailwind CSS IntelliSense, Prisma
- **Database Management**: pgAdmin, DBeaver, or Prisma Studio
- **API Testing**: Postman, Thunder Client, or curl
- **Container Management**: Docker Desktop GUI

Testing Frameworks Integration

This application is designed to work seamlessly with:

- **Selenium WebDriver**: Java, Python, C#, JavaScript
- **Cypress**: JavaScript/TypeScript end-to-end testing
- **Playwright**: Multi-browser automation
- **WebDriverIO**: Node.js testing framework
- **TestCafe**: JavaScript testing without WebDriver

Next Steps for Test Implementation

1. Choose your preferred testing framework
2. Set up Page Object Model structure
3. Create base page classes
4. Implement page-specific classes
5. Write comprehensive test suites
6. Set up CI/CD integration
7. Generate test reports

This comprehensive guide provides everything needed to get started with Page Object Model testing using a modern, full-stack web application. The application includes real-world complexity while maintaining clean, testable architecture.