## General E-Commerce MarketPlace Project : Full Documentation ( Day 1-6)

#### Day 01 : General E-commerce

is the buying and selling of goods and services online. Its primary purpose is to provide convenience to consumers by allowing them to shop from anywhere, while helping businesses reach a larger audience and streamline their sales process.

#### Target Audience for E-commerce :

#### Tech-Savvy Shoppers :

People who are comfortable with technology and prefer the convenience of shopping from their phones or computers, often searching for quick purchases, deals, or specific products.

#### Busy Individuals :

Those with tight schedules who want to save time by shopping online instead of visiting physical stores. This includes professionals, parents, and students.

#### Global Consumers :

E-commerce breaks geographical barriers, so it targets customers worldwide, offering access to international products and services that they might not find locally.

#### Price-Sensitive Shoppers :

People who compare prices easily online, looking for discounts, offers, or the best deals available in the market.

#### Specific Niche Groups:

E-commerce platforms often cater to particular interest groups, like fitness enthusiasts, gamers, or fashion lovers, offering specialized products tailored to their needs.

# Day 02 : Technical Planning Documentation

#### Overview

This document presents the technical plan for developing an E-Commerce Marketplace aimed at empowering small businesses and individuals by offering a platform for selling their products online. The plan builds upon the ideas generated during Hackathon Day 1 and integrates the recommendations outlined in the Day 2 guidelines.

#### Key Technologies

· **Frontend Framework**: Next.js

· **Content Management System (CMS)**: Sanity

· **Order Management and Shipping Integration**: ShipEngine

· **Database Solution**: MongoDB (for user authentication and data storage)

· **Hosting and Deployment**: Vercel (for frontend hosting), AWS (for backend infrastructure)

· **Payment Gateway Integration**: Stripe

## Day 03: API Integration & Data Migration

## Introduction

Shopify is a popular eCommerce platform that allows users to build online stores and sell products. Shopify provides an API that lets you fetch store data such as products, collections, and orders. By integrating Shopify's API into your **Next.js** project, you can display dynamic content such as products and collections on your website. **Next.js** provides server-side rendering (SSR) and static site generation (SSG), making it a great choice for building fast, SEO-friendly web applications.

Step 1: Shopify Store Setup  
  
1.1 Create a Shopify Store  
  
 If you haven't already, you'll need to create a Shopify store first.

 Visit [Shopify](https://www.shopify.com" \t "_new) and sign up.

 Set up your store with a name, theme, and products.

 Log in to your Shopify Admin panel.

## 1.2 Create a Private App

To access Shopify’s API, you need to create a private app that provides an API key and password.

 In the Shopify Admin panel, go to Apps.

 Click on Manage private apps and then Create a new private app.

 Provide an app name and set the required API permissions (e.g., Products, Orders, Inventory).

 Once the app is created, you’ll be provided with an API Key and API Password. These credentials will be used to authenticate API requests.

##### Day 04 : Details documentation for dynamic components and functionalities

Building a dynamic marketplace with Next.js and Shopify allows you to create a fast, scalable, and interactive e-commerce platform. This process involves integrating dynamic components such as product listings, cart management, user authentication, and real-time data fetching to deliver a seamless shopping experience. In this guide, I'll walk you through the essential components and functionalities needed to build a professional marketplace, with detailed documentation to help you implement each feature efficiently.

## Dynamic Components in Next.js

What Are Dynamic Components?

Dynamic components are those that load their content or behavior at runtime, meaning their data can change based on user interactions, external data sources, or other factors. In the case of a marketplace, products, pricing, and cart data are dynamic as they change frequently.

Next.js offers several tools to work with dynamic components, including:

* **Dynamic Routes**
* **Server-Side Rendering (SSR)**
* **Static Site Generation (SSG)**
* **Incremental Static Regeneration (ISR)**
* **API Routes**

## Day 5 : Testing, Error Handling, and Backend Integration Refinement

On Day 5, the focus is on ensuring the marketplace operates smoothly by thoroughly testing its components, handling errors gracefully, and refining the integration between the frontend (Next.js) and backend (Shopify). This process helps in improving the reliability, performance, and user experience of the marketplace**.**

## Testing Dynamic Components and Functionality

Testing ensures that the application behaves as expected under various conditions. This includes unit tests for individual components, integration tests for ensuring seamless interaction between different parts, and end-to-end (E2E) tests for the full user experience.

## Key Testing Strategies:

* Unit Testing: Test small parts of the application like individual React components or utility functions.
* Integration Testing: Ensure that the various parts of the system (frontend, API calls, and external services like Shopify) work together as expected.
* End-to-End Testing: Test user flows such as product search, adding items to the cart, and completing the checkout process.

## CSV-Based Testing Report

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Description** | **Test steps** | **Expected Result** | **Actual result** | **Status** | **Severity** | **Remarks** |
| TC\_001 | Verify product listing  Loads | Go to product page and check product listing | All products should be listed  correctly | All products are listed correctly | Passed | High | No issues |
| TC\_002 | Verify product details  page | Click on a product | it should open the product detail  page | Product detail page should open correctly | Product detail page  opened correctly | Passed | Team B |
| TC\_003 | Verify cart  functionality | Add product to cart and check cart | Cart should show the correct  Number of items | Cart updated correctly | Passed | High | No issues |
| TC\_004 | Verify checkout  process | Complete the checkout process | User should be able to complete  checkout without errors | Checkout completed successfully | Passed | Critical | No issues |
| TC\_005 | Verify mobile  responsiveness | Check marketplace on mobile devices | Marketplace should display  correctly on mobile | Marketplace displays correctly on mobile | Passed | Medium | No issues |
| TC\_006 | Verify error  Handling on API failure | Simulate API failure | Error message should be shown to  User | Error message displayed correctly | Passed | High | Handled gracefully |
| TC\_007 | Verify user login | Login with valid credentials | User should be able to log in  successfully | User logged in successfully | Passed | High | No issues |
| TC\_008 | Verify product search | Search for a product | Search should return accurate  Results | Search results are accurate | Passed | Medium | No issues |
| TC\_009 | Verify SEO tags | Check HTML page source for SEO tags | SEO tags should be present in the  source code | SEO tags present correctly | Passed | High | No issues |
| TC\_010 | Verify accessibility  score | Check accessibility score using tools | Accessibility score should be 100 | Accessibility score 100 | Passed | High | No issues |

## CSV Content

Test case id,test case description,test steps,expected result,actual result,status,severity,assigned to,remarks

TC\_001,Verify product listing loads,Go to product page and check product listing,All products should be listed correctly,All products are listed correctly,Passed,High,Team A,No issues

TC\_002,Verify product details page,Click on a product, it should open the product detail page,Product detail page should open correctly,Product detail page opened correctly,Passed,High,Team B,No issues

TC\_003,Verify cart functionality,Add product to cart and check cart,Cart should show the correct number of items,Cart updated correctly,Passed,High,Team A,No issues

TC\_004,Verify checkout process,Complete the checkout process,User should be able to complete checkout without errors,Checkout completed successfully,Passed,Critical,Team C,No issues

TC\_005,Verify mobile responsiveness,Check marketplace on mobile devices,Marketplace should display correctly on mobile,Marketplace displays correctly on mobile,Passed,Medium,Team D,No issues

TC\_006,Verify error handling on API failure,Simulate API failure,Error message should be shown to user,Error message displayed correctly,Passed,High,Team B,Handled gracefully

TC\_007,Verify user login,Login with valid credentials,User should be able to log in successfully,User logged in successfully,Passed,High,Team A,No issues

TC\_008,Verify product search,Search for a product,Search should return accurate results,Search results are accurate,Passed,Medium,Team D,No issues

TC\_009,Verify SEO tags,Check HTML page source for SEO tags,SEO tags should be present in the source code,SEO tags present correctly,Passed,High,Team C,No issues

TC\_010,Verify accessibility score,Check accessibility score using tools,Accessibility score should be 100,Accessibility score 100,Passed,High,Team D,No issues

#### Day 6 : Deployment Preparation & Staging Environment Setup

Day 6 focuses on preparing the marketplace for deployment by setting up a staging environment that mirrors the production environment. This will ensure that all components, including integrations with Shopify, are fully functional before we push the changes live.

#### Key Highlights:

#### Staging Environment Setup:

  Configured a separate staging environment to mirror the production setup, allowing for thorough testing before deployment.

  Ensured Shopify integration, from product fetching to checkout, works seamlessly in the staging environment.

#### Deployment Pipeline Configuration:

* Set up CI/CD pipelines using tools like GitHub Actions or Jenkins for automated testing and deployment.
* Integrated the pipeline to automatically deploy to staging and notify the team of any errors.

Security and Backu**p**:

· Secured environment variables and API keys in staging using environment management tools.

· Set up automated backups to ensure data integrity before any updates are made.

#### Domain and SSL Configuration:

· Configured custom domains for both staging and production environments.

· Ensured SSL certificates are set up for secure communication over HTTPS in both environments.

#### Shopify Integration Testing:

· Tested Shopify's Storefront API for product listings, cart functionality, and checkout to ensure smooth operation.

· Validated Shopify Payments and Checkout functionality within the staging environment.

#### Performance Optimization:

· Conducted performance tests in staging, focusing on speed and concurrency handling.

· Optimized images and JavaScript files to enhance loading time.

#### GitHub Repository Structure

my-marketplace/

│

├── .github/ # GitHub-specific configurations (e.g., CI/CD workflows)

│ ├── workflows/ # GitHub Actions or other CI/CD pipelines

│ │ ├── deploy.yml # Deployment pipeline

│ │ └── test.yml # Test pipeline

│

├── assets/ # Static assets like images, logos, etc.

│ ├── images/

│ ├── icons/

│ └── fonts/

│

├── components/ # Reusable UI components

│ ├── Cart.js

│ ├── ProductCard.js

│ ├── Header.js

│ └── Footer.js

│

├── config/ # Configuration files for APIs, settings, etc.

│ ├── shopify.js # Shopify Storefront API config

│ ├── seo.js # SEO-related configuration

│ └── env.js # Environment variables (local & production settings)

│

├── pages/ # Next.js pages (for dynamic routing)

│ ├── \_app.js # Main app setup

│ ├── \_document.js # Document customization

│ ├── index.js # Home page

│ ├── products/ # Dynamic routing for product pages

│ │ └── [id].js # Individual product detail page (dynamic route)

│ ├── cart.js # Cart page

│ ├── checkout.js # Checkout page

│ └── api/ # API routes

│ ├── cart.js # Cart API functions

│ └── checkout.js # Checkout API functions

│

├── public/ # Public folder for static assets served directly

│ ├── images/

│ ├── favicon.ico

│ └── robots.txt

│

├── styles/ # Global styles and CSS modules

│ ├── globals.css # Global CSS

│ ├── Home.module.css # Home page-specific styles

│ └── Cart.module.css # Cart page-specific styles

│

├── utils/ # Utility functions (helpers, hooks)

│ ├── fetchShopify.js # Helper to fetch data from Shopify

│ ├── formatCurrency.js # Helper to format product prices

│ └── useCart.js # Custom hook for cart management

│

├── .env # Environment variables (Shopify API keys, etc.)

├── next.config.js # Next.js configuration

├── package.json # Project metadata and dependencies

├── README.md # Project overview and documentation

└── yarn.lock # Yarn lockfile (if using Yarn)

#### Conclusion:

Day 6 was dedicated to ensuring that the marketplace is fully prepared for a smooth deployment. With a fully configured staging environment, continuous integration and deployment pipelines, and Shopify integration fully tested, we are ready to proceed with the final production deployment. This ensures that our marketplace is ready for a real-world launch.