# -Store Short Report

## 1. Rendering Choices and Rationale

|  |  |  |
| --- | --- | --- |
| Page / Route | Rendering Strategy | Rationale |
| / | SSG (Static Site Generation) | Homepage content is mostly static (marketing, featured products) and doesn’t change often. Using SSG ensures fast load times and excellent SEO. |
| /dashboard | CSR (Client-Side Rendering) | Dashboard content is personalized for each admin/user. CSR allows fetching live data dynamically after initial page load. |
| /products/[slug] | SSR (Server-Side Rendering) | Product details (pricing, stock, description) update frequently. SSR ensures that users always see the latest data and improves SEO for product pages. |
| /admin | CSR (Client-Side Rendering) | Admin dashboard interacts heavily with API for CRUD operations. CSR provides instant interactivity and fetches data dynamically from the backend. |

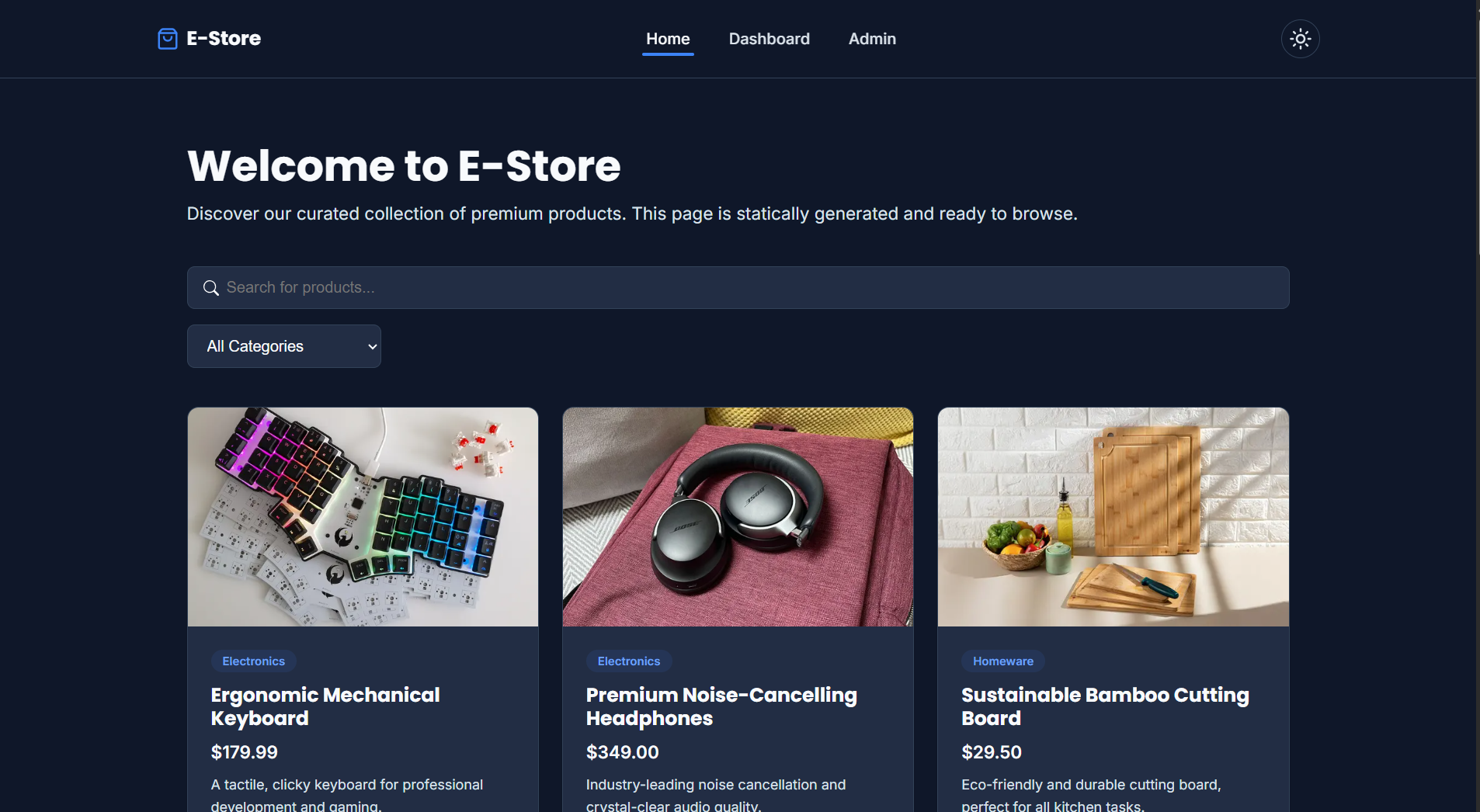
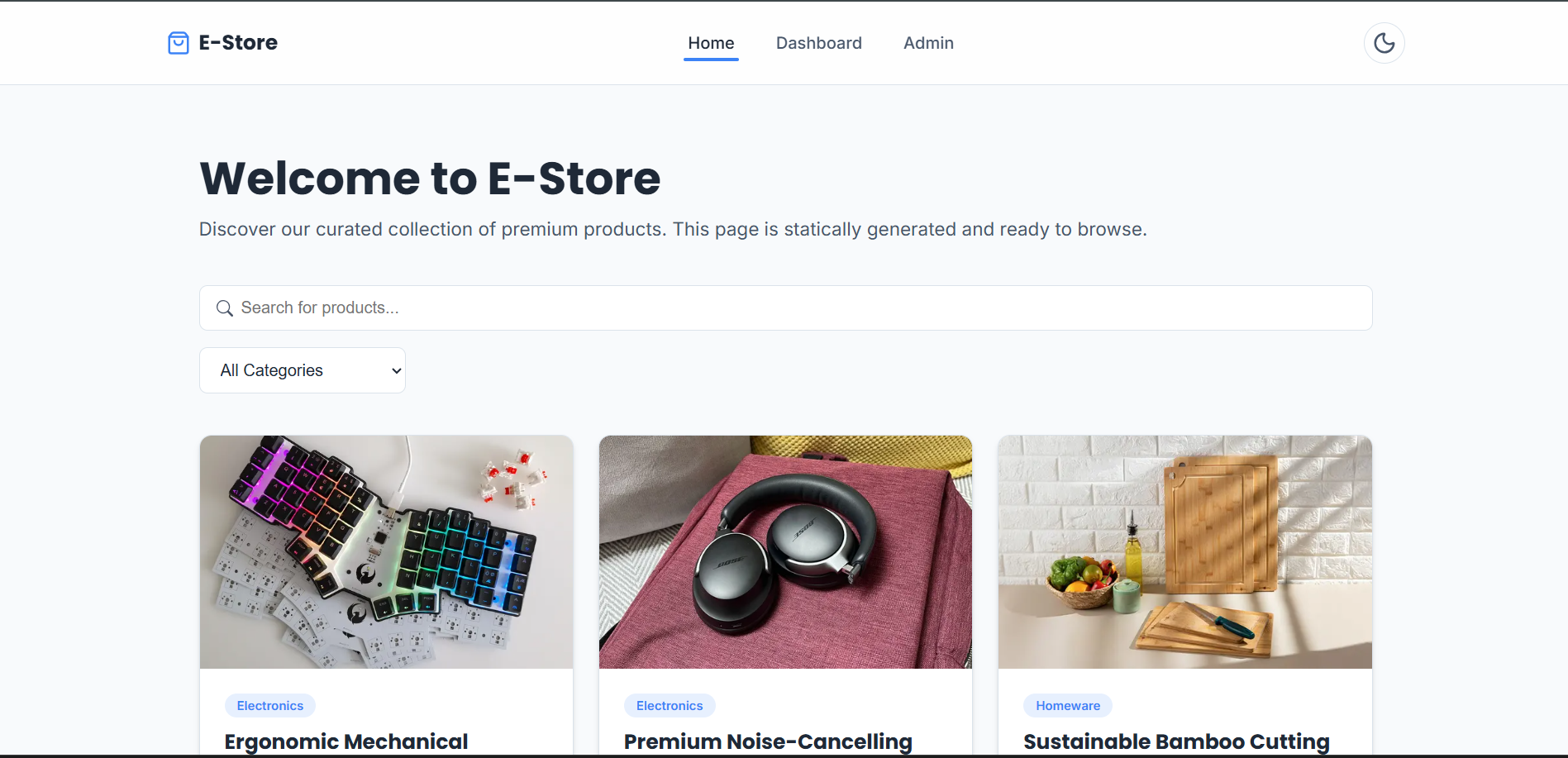
## 2. Data Flow Between Frontend and Backend

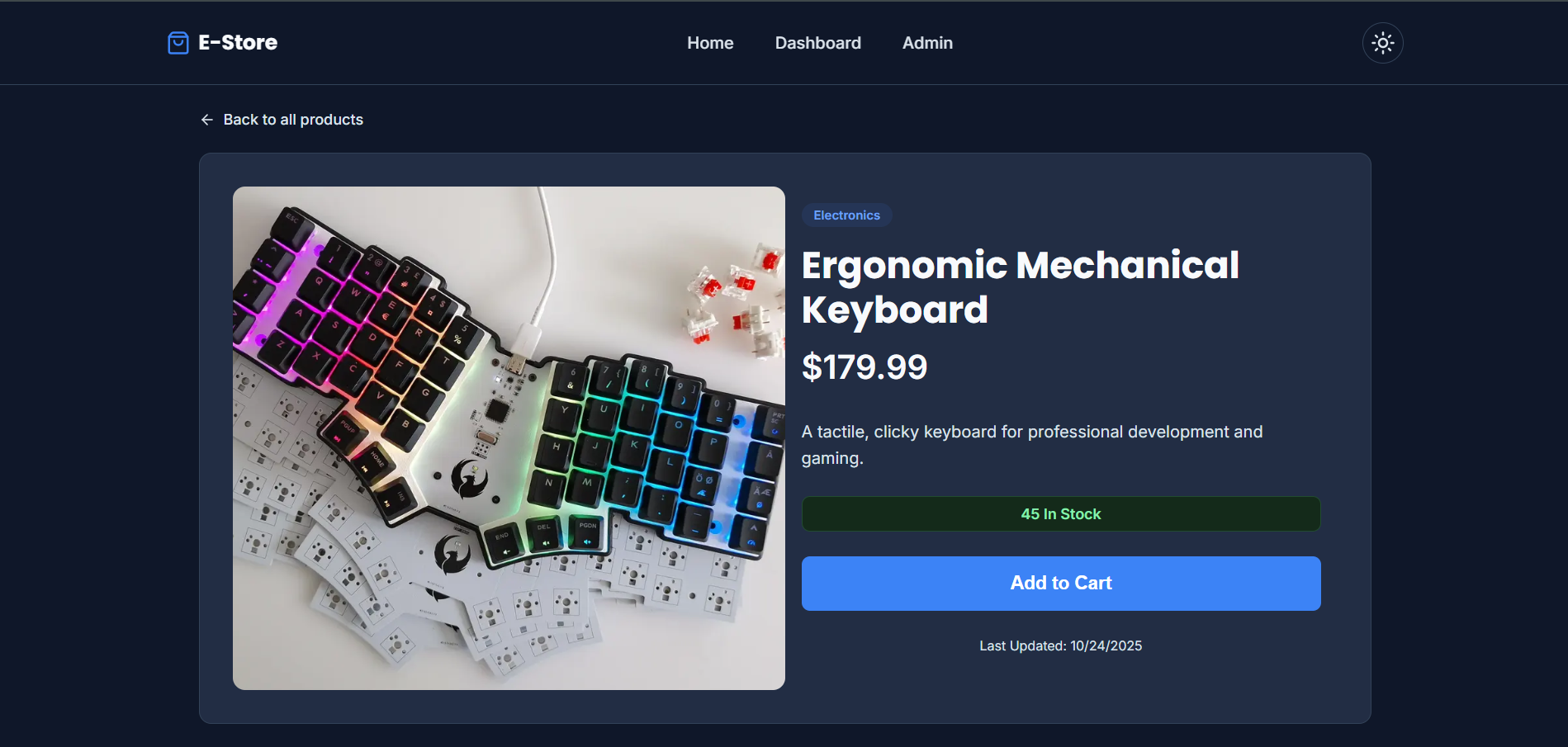
1. Frontend (Next.js App Router / Admin UI)  
- Components fetch data using `fetch` calls to API routes (e.g., `/api/products`)  
- Admin forms (add/update/delete) send JSON payloads to API routes with authentication headers.  
  
2. Backend (API Routes / MongoDB)  
- API routes handle requests for CRUD operations using helper functions (`getProductBySlug`, `updateProductBySlug`, etc.)  
- MongoDB stores products, inventory, and other data persistently.  
- Data returned from APIs is sent as JSON to the frontend.  
  
3. Example Flow:  
- Admin updates product → Frontend sends `PUT /api/products/[slug]` → Backend updates MongoDB → Frontend fetches updated list → UI refreshes dynamically.

## 3. Challenges Faced & Solutions

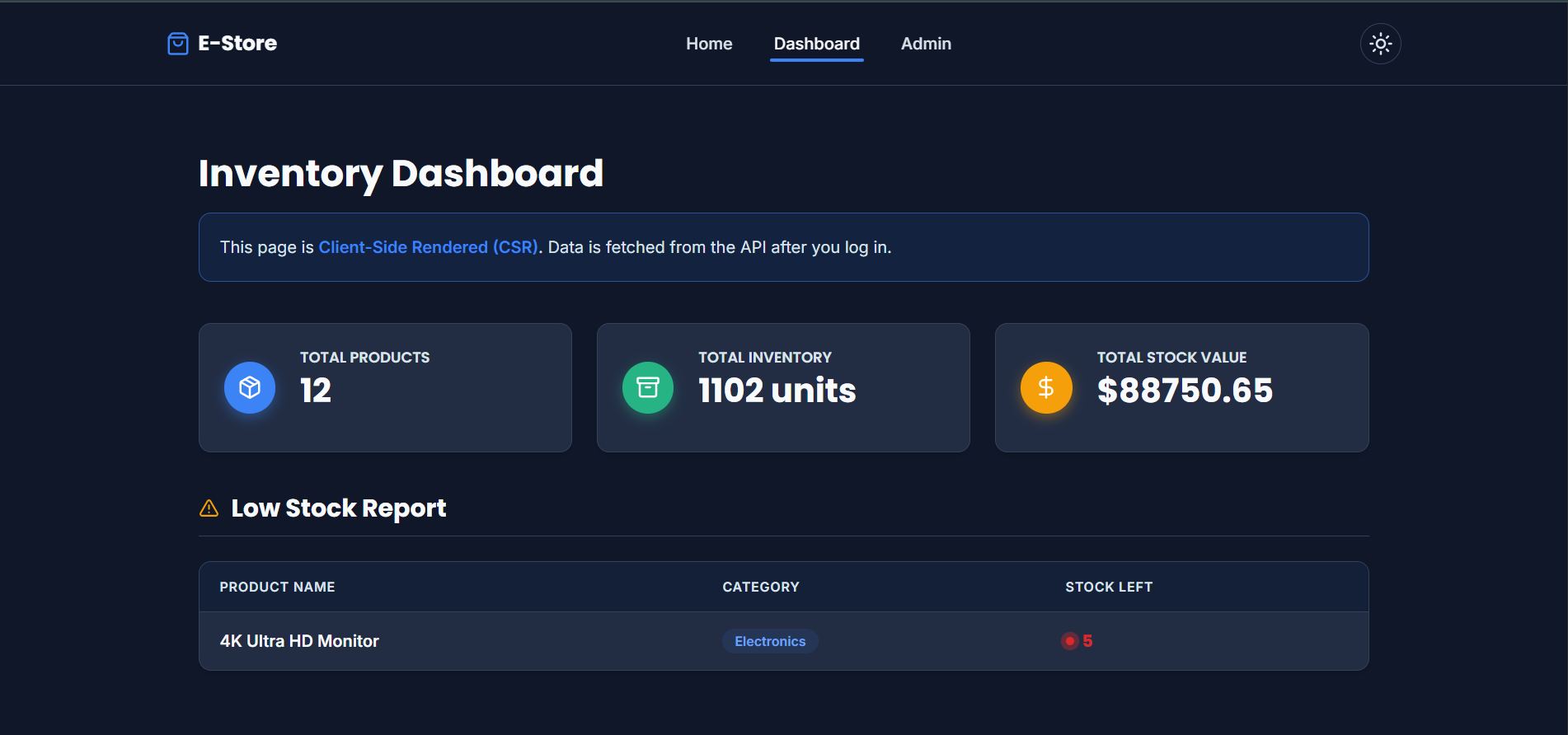
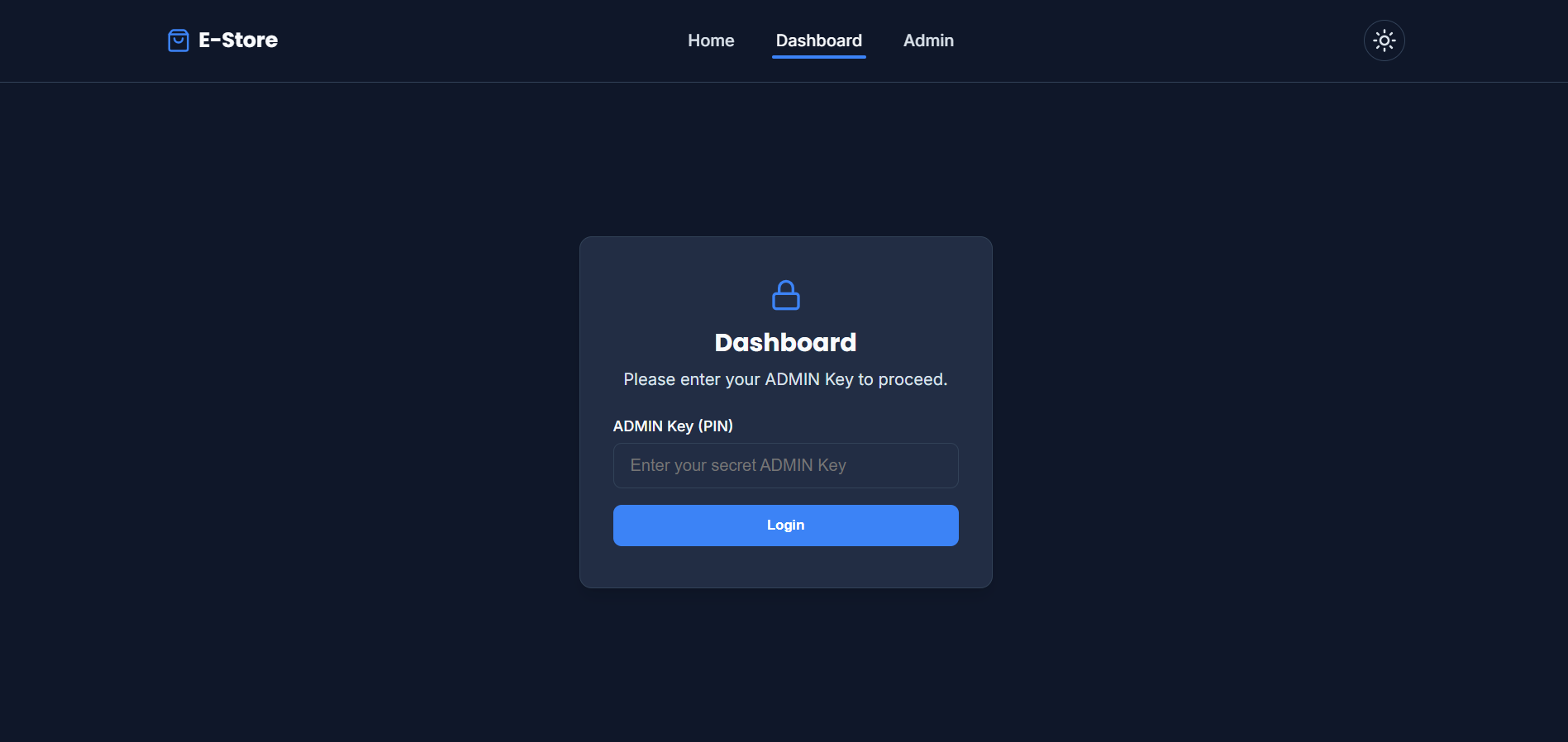
- Dynamic Routes in App Router:  
 - Problem: `params` in API routes returned as a Promise.  
 - Solution: Unwrapped `params` using `await` to properly fetch the slug.  
  
- Client-Side Rendering vs Static Rendering:  
 - Problem: Admin dashboard required real-time data but some pages needed static generation.  
 - Solution: Applied CSR for admin/dashboard and SSG/SSR/ISR strategically elsewhere.  
  
- Favicon and Images:  
 - Problem: Initial favicon changes didn’t apply; Unsplash links sometimes inconsistent.  
 - Solution: Converted SVG to proper favicon file and optionally used Unsplash API with access key for consistent images.  
  
- API Security:  
 - Problem: Unauthorized access to API routes could occur.  
 - Solution: Implemented API Key check in headers for admin operations.

## 4. Screenshots

(Include 1–2 screenshots per page. Suggested order: Homepage, Product Page, Admin Dashboard, Dashboard Page)  
- Homepage (SSG) 

-Products page

-Dashboard page

-Admin Page

