8

**COLLEGE CODE**: 9623

**COLLEGE NAME**: AMRITA COLLEGE OF ENGINEERING AND TECHNOLOGY

**DEPARTMENT**: COMPUTER SCIENCE AND ENGINEERING

**STUDENT NM-ID**

DCF4FDCE8362E5386C147E8DD45A8E24

**ROLL NO**: 962323104006

**DATE**: 12-09-2025

COMPLETED THE PROJECT NAMED AS PHASE 2

# E-Commerce Product Page — Architecture & Design

**SUBMITTED BY,**

**NAME**: P.S.Abiram

**MOBILE.NO**:8189846996

E-Commerce Product Page — Architecture & Design

- **Tech Stack Selection**

- Frontend
- Next.js + React + TypeScript→ SEO (SSR/SSG), developer productivity.
- Tailwind CSS → utility-first styling.
- React Query / SWR → caching, background revalidation.
- Vercel / Cloudflare Pages → edge SSR, CDN caching.

- Backend

- Node.js + NestJS → modular, testable API layer.
- GraphQL or REST → GraphQL if client flexibility is needed.
- PostgreSQL for product catalog, Elasticsearch for search.

❖ Redis for caching sessions & inventory counters.

➢ Stripe / Adyen for payments.

❖ Observability: Prometheus + Grafana, Sentry, OpenTelemetry.

❖ 3. UI Structure (Component Hierarchy)

❖ *Page Layout

- Header (logo, search, account, cart).
- Breadcrumbs + SEO metadata.
- Product Gallery (images, zoom).
- Product Info (title, price, stock, CTA buttons).
- Tabs/Accordions: Description, Specs, Reviews.

- Recommendation Carousel.
- Footer (links, policies).

❖ Components: ProductGallery, VariantSelector, AddToCartButton, Reviews, RecommendationsCarousel, Breadcrumbs, StockIndicator.

❖ ---

**API Schema & Data Handling**

❖ 4. **API Schema Design (REST Example)**

❖ GET /api/products/\:id

❖ json
❖ {
❖ "id": "sku_123",
❖ "title": "Acme Running Shoes",
❖ "price": { "currency": "USD", "amount": 119.99 },
❖ "variants": [{ "id":"v1","color":"red","size":"9" }],
❖ "images": [{ "url":"https://cdn/.../img1.avif", "alt":"side view" }],
❖ "rating": { "average": 4.5, "count": 238 }
❖ }

❖ POST /api/cart → add product to cart, returns updated cart.
❖ GET /api/products/\:id/reviews→ paginated reviews.
❖ GET /api/recommendations→ related products.

❖ GraphQL Sketch

❖ graphql
❖ type Product {
❖ id: ID!
❖ title: String!
❖ price: Price!
❖ variants: [Variant!]
❖ images: [Image!]
❖ rating: Rating
❖ }

❖ type Query {
❖ product(id: ID!): Product
❖ recommendations(productId: ID!): [Product!]
❖ }

5. Data Handling Approach

❖ Client-side

- ➢ React Query cache + stale-while-revalidate.
- ➢ Optimistic UI for Add-to-Cart.
- ➢ LocalStorage + server sync for cart.

❖ Server-side

- ➢ Postgres as source of truth.

- Elasticsearch for search/facets.
- CDC pipeline to update search index.
- Redis for caching and session storage.

❖ Images & Media: AVIF/WebP responsive sizes, CDN delivery.

❖ Analytics: Stream events to Kafka → Snowflake/BigQuery.

**Diagrams & Flows**

6. **Component / Module Diagram**

❖ mermaid
❖ flowchart LR
❖ subgraph Frontend
❖ A[Next.js App] --> B[Product Components]
❖ B --> C[React Query Cache]
❖ end

❖ subgraph Edge
❖ D[BFF / Edge Functions] --> E[API Gateway]
❖ end

❖ subgraph Backend
❖ E --> F[Product Service]
❖ E --> G[Inventory Service]
❖ E --> H[Search Service]
❖ E --> I[Recommendations]

❖ E --> J[Reviews]
❖ F --> S3[(Images)]
❖ end

7. **Basic Flow Diagrams**

❖ A. Page Load (SSR)

❖ mermaid
❖ sequenceDiagram
❖ U->>CDN: Request Product Page
❖ CDN->>App: SSR Render
❖ App->>API: Fetch product data
❖ API-->>App: JSON response
❖ App->>CDN: HTML with product data
❖ CDN->>U: Deliver page
❖ U->>App: Hydration + fetch incremental data

❖ B. Add to Cart

❖ mermaid
❖ sequenceDiagram
❖ U->>FE: Click Add
❖ FE->>FE: Optimistic UI update
❖ FE->>API: POST /cart
❖ API->>Inventory: Reserve stock
❖ Inventory-->>API: Confirm/Reject

- API-->>FE: Updated cart / rollback

- 8. Operational Concerns

  - CDN cache by SKU + locale.
  - Read replicas & materialized views for performance.
  - Feature flags for A/B tests.
  - Rate limiting on write endpoints.

- 9. Next Steps

  - Create Figma wireframes.
  - Generate OpenAPI or GraphQL SDL.
  - Scaffold Next.js ProductPage component with Tailwind.