Farm2door E-Commerce Website
Project Documentation

Interim Project

Group Project

Team Members:

1. **Nitish Kumar (2461278)**
2. **Preety Chaurasiya (2461416)**
3. **Swathy R(2461480)**
4. **Rakshitha R(2461277)**

1. Title & Overview

Project Name and Version: Farm2door E-Commerce Website

Short Description: Farm2door is an online shopping platform where users can browse products, search and filter categories, add items to a wishlist and cart, and proceed to checkout. Registered users can manage their personal profile and addresses, while administrators maintain the catalog through an Admin Dashboard.

Purpose and Scope: This document explains the application's design, features, architecture, user flows, components, contexts/APIs, installation steps, testing, security considerations, and future improvements. It is intended for stakeholders, developers, testers, and admins who need a clear understanding of how the system works.

2. Introduction

Background: Farm2door was created to simplify online purchasing by providing an intuitive interface for discovering and buying products. It supports both end users and administrators, ensuring the catalog stays accurate and up to date.

Objectives & Goals: Deliver a responsive storefront, secure authentication, easy wishlist/cart management, straightforward checkout, and a robust admin interface for product management and order visibility.

Target Audience: End users (shoppers), administrators (catalog managers), and developers/testers maintaining or extending the platform.

Business Problem: Many small catalogs lack a unified, secure and easy-to-use system. Farm2door streamlines browsing, ordering, and catalog maintenance, reducing overhead and improving user satisfaction.

3. System Architecture

High-level Design: The application uses a client-side SPA for the storefront and an admin UI. Authentication gates user actions (wishlist, profile, checkout) and role-based access controls admin features. Data is persisted locally for demo purposes and can be connected to a backend for production.

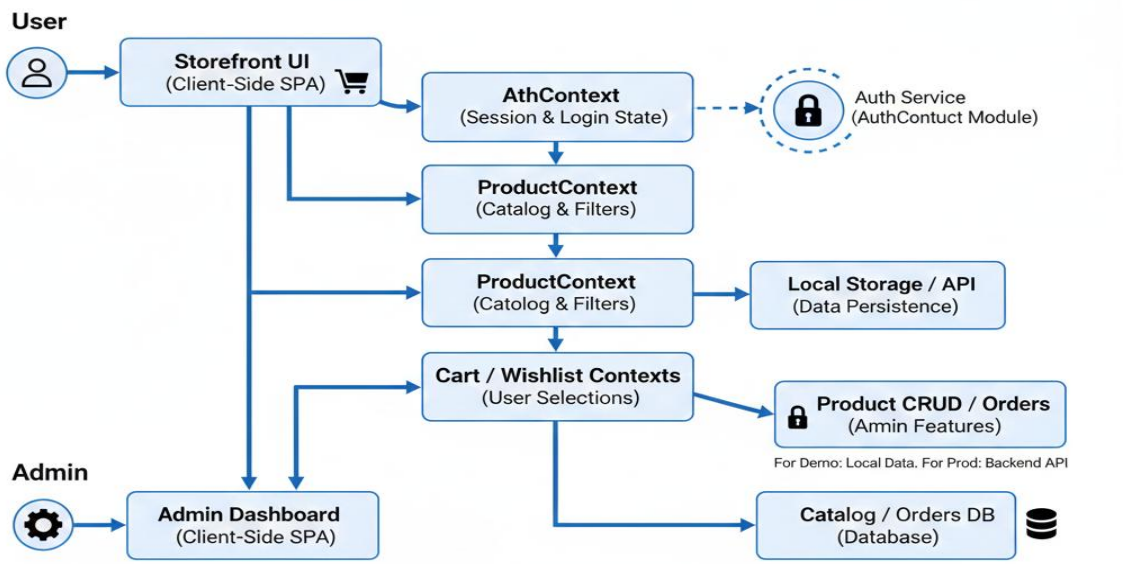
Technologies Used: Frontend (HTML/CSS/JavaScript or React), State Management via Context, Optional Backend (Node.js/Express), Database (MySQL/MongoDB), Authentication (JWT or session).

Contexts/Modules: AuthContext for login/logout and session state; ProductContext (or MovieContext if codebase uses that name) for catalog items, filters, and mutations like add/update/delete; Cart/Wishlist contexts for user selections.

High-level Interaction Diagram:

User → Storefront UI → AuthContext ——→ Auth Service (JWT/session)
→ ProductContext → Catalog (products, categories)
→ Wishlist/Cart Context → Local storage / API
Admin → Admin Dashboard → Product CRUD / Orders → Catalog / Orders DB

System Architecture: High-Level Interaction Diagram




```
// src/App.jsx
import React, { useContext } from 'react'
import { BrowserRouter, Routes, Route, Navigate, useLocation } from
'react-router-dom'

import { AuthProvider, AuthContext } from './context/AuthContext'
```

```jsx
import { ProductProvider } from './context/ProductContext'
import { ToastProvider } from './context/toastcontext' // NEW: Toasts
provider

import Header from './components/Header'
import BottomNav from './components/BottomNav'

import Home from './pages/Home'
import CategoryList from './pages/CategoryList'
import Wishlist from './pages/Wishlist'
import Account from './pages/Account'
import Login from './pages/Login'
import AdminDashboard from './pages/AdminDashboard'
import Profile from './pages/Profile'
import Cart from './pages/Cart'
import OrderHistory from './pages/OrderHistory'
import ProductDetail from './pages/ProductDetail'

// Route guard: requires login
function RequireAuth({ children }) {
  const { isLoggedIn } = useContext(AuthContext)
  const location = useLocation()

  if (!isLoggedIn) {
    // Redirect to login and remember where user was going
    return <Navigate to="/login" state={{ from: location }} replace />
  }
  return children
}

export default function App() {
  return (
    <BrowserRouter>
      <AuthProvider>
        <ProductProvider>
          <ToastProvider>{/* ⬅ wrap the whole app so toasts work
everywhere */}
            <div className="app-root">
              <Header />
              <main className="app-main">
                <Routes>
                  {/* Public routes */}
                  <Route path="/" element={<Home />} />
                  <Route path="/category/:category" element={<CategoryList
/>} />
```

```
                <Route path="/category" element={<CategoryList />} />
                <Route path="/login" element={<Login />} />
                <Route path="/admin" element={<AdminDashboard />} />
                <Route path="/product/:id" element={<ProductDetail />}
/>

                {/* Protected routes */}
                <Route
                  path="/wishlist"
                  element={
                    <RequireAuth>
                      <Wishlist />
                    </RequireAuth>
                  }
                />
                <Route
                  path="/account"
                  element={
                    <RequireAuth>
                      <Account />
                    </RequireAuth>
                  }
                />
                <Route
                  path="/cart"
                  element={
                    <RequireAuth>
                      <Cart />
                    </RequireAuth>
                  }
                />
                <Route
                  path="/orders"
                  element={
                    <RequireAuth>
                      <OrderHistory />
                    </RequireAuth>
                  }
                />
              </Routes>
            </main>

            {/* Drawer mounted once */}
            <Profile />
            <BottomNav />
```

```
            </div>
          </ToastProvider>
        </ProductProvider>
      </AuthProvider>
    </BrowserRouter>
  )
}
```

4. Features & Functionality

- User registration and login
- Search bar and category filters
- Wishlist management (add/remove)
- Cart and quantity updates
- Checkout workflow
- Profile with personal info and saved addresses
- Admin dashboard for product add/edit/delete and viewing orders

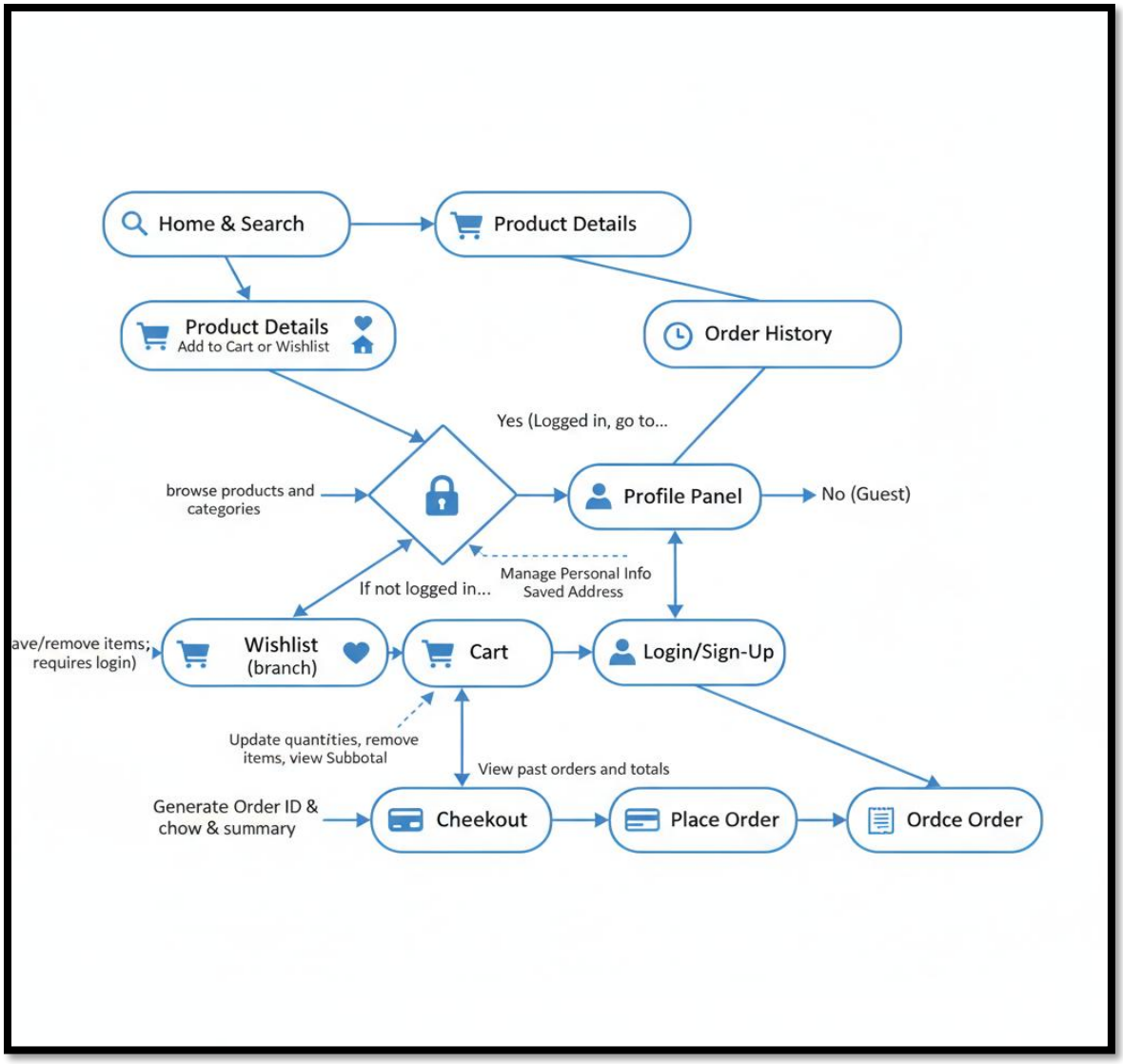
Role-based Features: Users can manage wishlist, cart, profile, and orders. Admins can add, edit, delete products, reset catalog, and view user/order lists.

Limitations/Constraints: Demo uses local storage and mock data in places; payment gateway and real backend can be integrated later. Performance depends on device and data size.

5. User Flow

User Journey: 1) Browse Home → 2) View product details → 3) Add to wishlist/cart (prompts login if not authenticated) → 4) Login/Sign-Up → 5) Manage profile and saved addresses → 6) Proceed to checkout → 7) View orders history.

Admin Journey: 1) Login as admin → 2) Open Admin Dashboard → 3) Add/Edit/Delete products → 4) Review users/orders → 5) Optionally reset catalog.



6. Component Documentation

HomePage

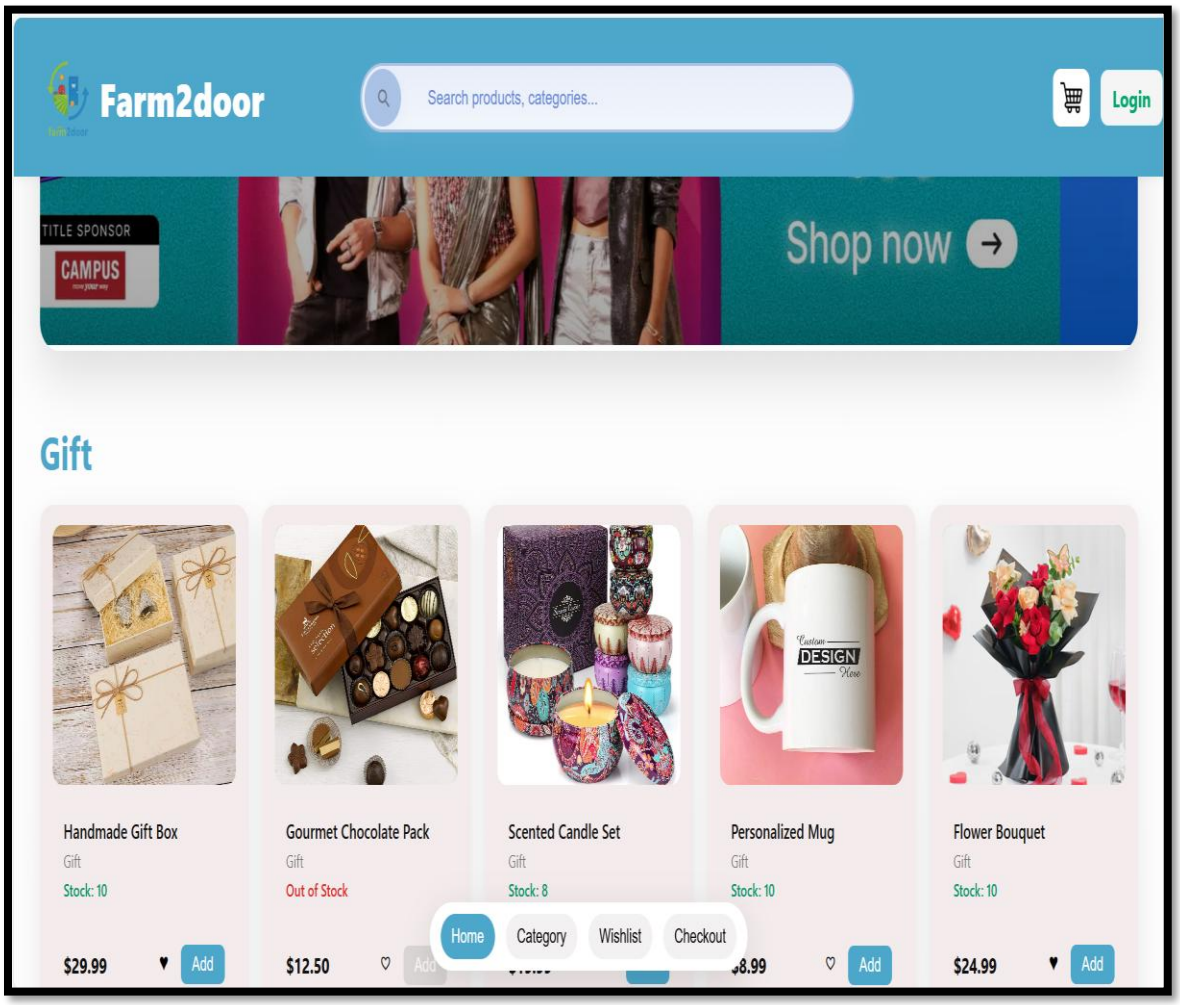
Purpose: Lists featured products and categories; hosts search bar.

Props/State: depends on framework; typically receives product/user objects and uses local state for form inputs.

Dependencies: AuthContext, ProductContext ,Router/Navigation, localStorage/API services.

Example Usage:

<HomePage /> // mounted within routes; interacts with contexts and services.



Code:

```
import React, { useContext, useEffect, useMemo } from 'react'
import Slider from 'react-slick'//after installing slick-carousel
import { useNavigate, useLocation } from 'react-router-dom'

import { AuthContext } from '../context/AuthContext'
import { ProductContext } from '../context/ProductContext'
import { useToast } from '../context/toastcontext'

import CategorySection from '../components/CategorySection'
import ProductCard from '../components/ProductCard'

// slick styles
import 'slick-carousel/slick/slick.css'
import 'slick-carousel/slick/slick-theme.css'
```

```
// banner images
import banner1 from '../assets/Banner/banner1.jpg'
import banner2 from '../assets/Banner/banner2.jpg'
import banner4 from '../assets/Banner/banner4.jpg'
import banner5 from '../assets/Banner/banner5.jpg'

export default function Home() {
  const { filtered, setSearchTerm, addToCart, addToWishlist } =
useContext(ProductContext)
  const { isLoggedIn } = useContext(AuthContext)
  const { showToast } = useToast()

  const navigate = useNavigate()
  const location = useLocation()

  useEffect(() => {
    setSearchTerm('')
  }, [setSearchTerm])

  const categories = ['Gift', 'Beauty', 'Electronics', 'Fruits']
  const heroBanners = useMemo(() => [banner1, banner2, banner4, banner5],
[])

  /* HERO BANNER SETTINGS (slide) */
  const heroSettings = {
    dots: true,
    infinite: true,
    // turn off fade to enable sliding
    fade: false,
    speed: 600,               // slide transition speed
    slidesToShow: 1,
    slidesToScroll: 1,
    autoplay: true,
    autoplaySpeed: 3000,
    arrows: true,             // show arrows for manual slide
    pauseOnHover: true,
    draggable: true,          // mouse drag
    swipe: true,              // touch swipe
    swipeToSlide: true,
    cssEase: 'ease',          // easing for slide animation
    dotsClass: 'slick-dots hero-dots',
  }

  /* FEATURED SLIDER SETTINGS */
```

```
  const featuredSettings = {
    dots: false,
    infinite: true,
    speed: 450,
    slidesToShow: 5,
    slidesToScroll: 1,
    autoplay: true,
    autoplaySpeed: 2500,
    lazyLoad: 'ondemand',
    responsive: [
      { breakpoint: 1200, settings: { slidesToShow: 4 } },
      { breakpoint: 1024, settings: { slidesToShow: 3 } },
      { breakpoint: 768, settings: { slidesToShow: 2 } },
      { breakpoint: 480, settings: { slidesToShow: 1 } }
    ]
  }

  // Add to cart from Home: redirect to login if needed, otherwise add and
toast
  const handleAddFromHome = (product) => {
    if (!isLoggedIn) {
      navigate('/login', {
        state: {
          from: location,
          intent: { type: 'ADD_TO_CART', product }, // carry intent so we
can add after login
        },
      })
      return
    }
    addToCart(product)
    showToast('Item added to cart', 'success')
  }

  // Add to wishlist from Home
  const handleWishlistFromHome = (product) => {
    if (!isLoggedIn) {
      navigate('/login', {
        state: {
          from: location,
          intent: { type: 'TOGGLE_WISHLIST', product },
        },
      })
      return
    }
```

```
    addToWishlist(product)
    showToast('Item added to wishlist', 'success')
  }

  return (
    <div className="page-container">
      {/* //HERO BANNER    */}
      <div className="hero-banner">
        <Slider {...heroSettings}>
          {heroBanners.map((img, idx) => (
            <div key={idx} className="hero-slide">
              <img src={img} alt={`banner-${idx}`} loading="lazy" />
              <div className="hero-overlay" />
            </div>
          ))}
        </Slider>
      </div>

      {/* CATEGORY SECTIONS */}
      {categories.map((cat) => (
        <CategorySection
          key={cat}
          title={cat}
          products={filtered.filter(
            (p) => {
              // Handle both single category string and array of
categories
              const productCategories = Array.isArray(p.category) ?
p.category : [p.category];
              return productCategories.some(pc => (pc || '').toLowerCase()
=== cat.toLowerCase());
            }
          )}
          limit={6}
          // If your CategorySection renders ProductCard internally and
supports passing handlers,
          // you can thread these through (CategorySection ->
ProductCard). Otherwise, ProductCard's
          // own logic can handle login redirect + toasts.
          onAdd={handleAddFromHome}
          onWishlist={handleWishlistFromHome}
        />
      ))}

      {/* FEATURED PRODUCTS */}
```

```
      <section className="featured-section">
        <h2 className="section-title">Featured Products</h2>

        <Slider {...featuredSettings}>
          {filtered.slice(0, 15).map((product) => (
            <div key={product.id} className="featured-slide">
              <ProductCard
                product={product}
                useDialog={false}
                // Pass handlers in case your ProductCard accepts them.
                // If ProductCard doesn't, you can remove these props and
rely on ProductCard's internal logic.
                onAdd={handleAddFromHome}
                onWishlist={handleWishlistFromHome}
              />
            </div>
          ))}
        </Slider>
      </section>
    </div>
  )
}
```

LoginForm

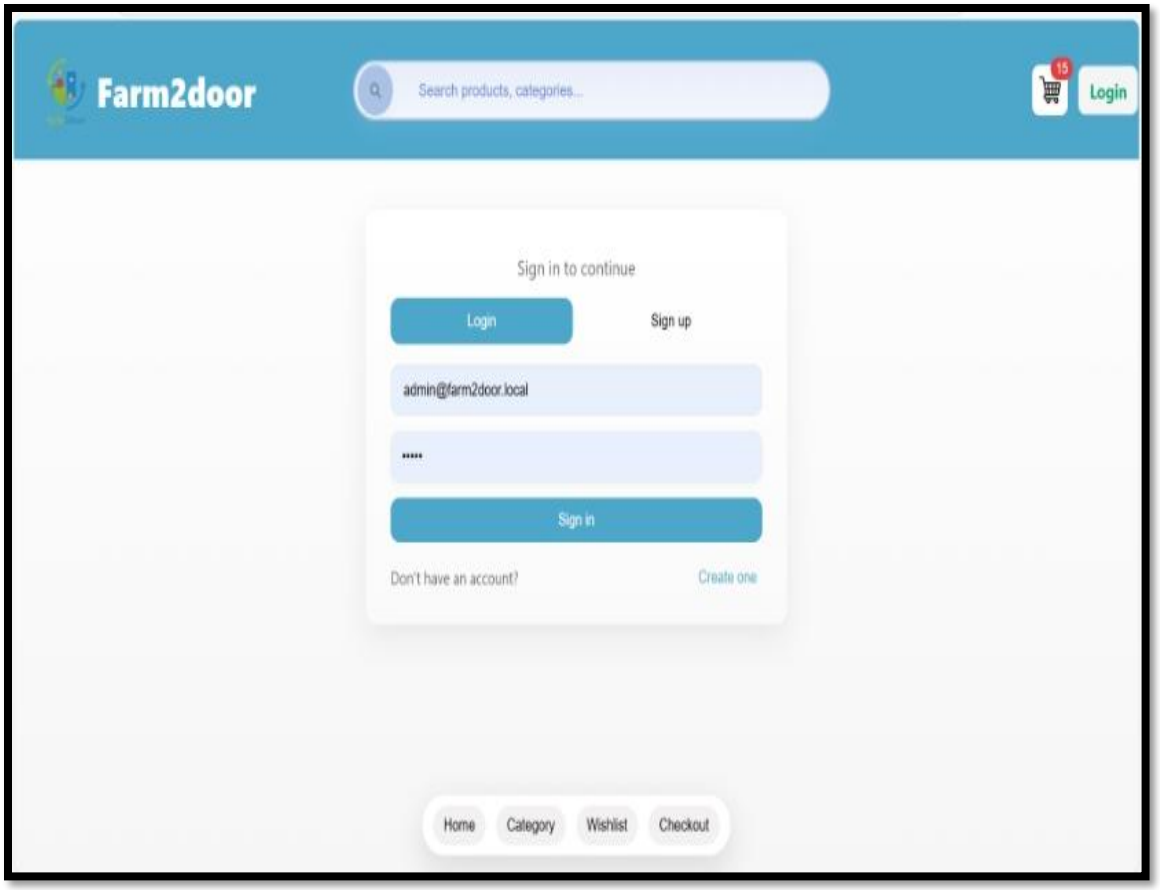
Purpose: Handles email/password input and submit; calls AuthContext.login.

Props/State: depends on framework; typically receives product/user objects and uses local state for form inputs.

Dependencies: AuthContext, ProductContext ,Router/Navigation, localStorage/API services.

Example Usage:

<LoginForm /> // mounted within routes; interacts with contexts and services



Login page – authenticate to continue

```
// src/pages/Login.jsx
import React, { useContext, useState } from 'react'
import { AuthContext } from '../context/AuthContext'
import { useNavigate, useLocation } from 'react-router-dom'
import { useToast } from '../context/toastcontext'

export default function Login() {
  const { login, signup } = useContext(AuthContext)
  const [mode, setMode] = useState('login') // 'login' | 'signup'
  const navigate = useNavigate()
  const location = useLocation()

  // Return to the page user came from (e.g., Home) after auth
  const from = location.state?.from?.pathname || '/'

  // feedback state
  const [error, setError] = useState('')
  const [success, setSuccess] = useState('')
```

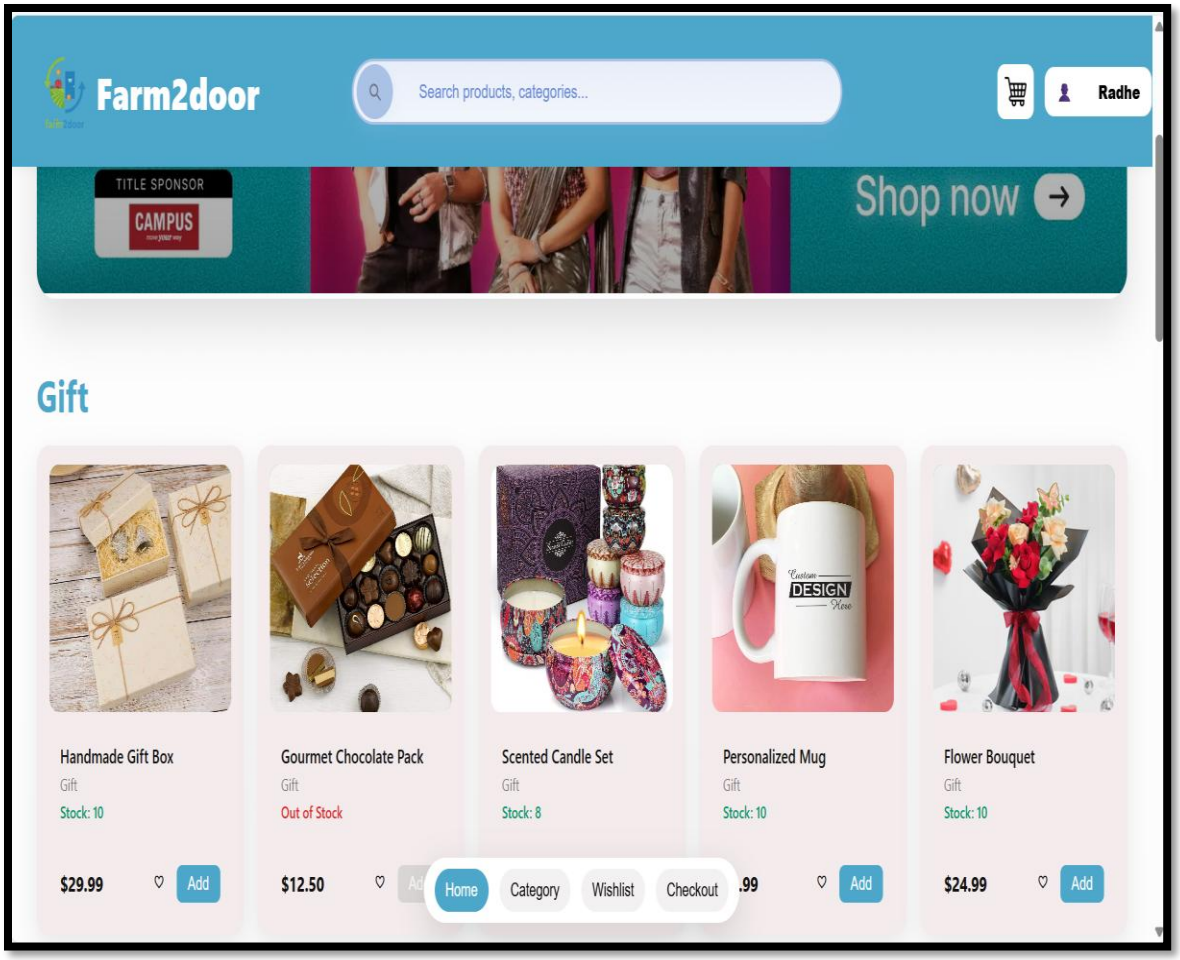
```
  // login state
  const [email, setEmail] = useState('')
  const [password, setPassword] = useState('')

  // signup state (avatar removed; phone now required)
  const [name, setName] = useState('')
  const [signupEmail, setSignupEmail] = useState('')
  const [signupPassword, setSignupPassword] = useState('')
  const [phone, setPhone] = useState('')

  // ========= PASSWORD VALIDATION (SIGNUP) =========
  // Min 8 chars, at least 1 uppercase, 1 lowercase, 1 number, 1 special
character
  function validatePassword(pw) {
    const strongRegex = /^(?=.*[a-z])(?=.*[A-
Z])(?=.*\d)(?=.*[^\w\s]).{8,}$/
    return strongRegex.test(pw)
  }
  const signupPasswordIsValid = validatePassword(signupPassword)

  // SIGNUP
  function handleSignup(e) {
    e.preventDefault()
    setError(''); setSuccess('')

    if (!name || !signupEmail || !signupPassword || !phone) {
      return setError('Please fill all required fields: Name, Email,
Phone, Password.')
    }

    // enforce strong password
    if (!signupPasswordIsValid) {
      return setError(
        'Password must be at least 8 characters and include uppercase,
lowercase, a number, and a special character.'
      )
    }

    // avatar removed; pass null/omit if your AuthContext expects it
    const res = signup({ name, email: signupEmail, password:
signupPassword, phone })
    if (!res?.ok) return setError(res?.message ?? 'Signup failed')

    setSuccess('Signup successful — you are logged in')
```

```
    navigate(from, { replace: true })
  }

  // LOGIN (unchanged strength; allow existing accounts)
  function handleLogin(e) {
    e.preventDefault()
    setError(''); setSuccess('')

    if (!email || !password) return setError('Enter email and password')

    const res = login({ email, password })
    if (!res?.ok) return setError(res?.message ?? 'Login failed')

    setSuccess('Login successful')
    navigate(from, { replace: true })
  }

  const errorStyle = { color: 'var(--danger)' }   // red message color
  const successStyle = { color: 'var(--accent)' } // success color

  return (
    <div className="page-container">
      <div className="auth-card">
        <h2 style={{ margin: 0, marginBottom: 6, textAlign: 'center' }}>
          {mode === 'signup'
            ? 'Create your account'
            : ''}
        </h2>

        <p style={{ marginTop: 0, marginBottom: 14, textAlign: 'center',
color: 'var(--muted)' }}>
          {mode === 'signup'
            ? 'Sign up to start ordering'
            : 'Sign in to continue'}
        </p>

        <div className="auth-tabs">
          <button
            className={`nav-btn ${mode === 'login' ? 'active' : ''}`}
            onClick={() => setMode('login')}
          >
            Login
          </button>
          <button
            className={`nav-btn ${mode === 'signup' ? 'active' : ''}`}
```

```
            onClick={() => setMode('signup')}
          >
            Sign up
          </button>
        </div>

        {/* LOGIN FORM */}
        {mode === 'login' && (
          <form onSubmit={handleLogin} className="auth-form">
            {error && <div style={errorStyle}>{error}</div>}
            {success && <div style={successStyle}>{success}</div>}

            <input
              aria-label="Email"
              placeholder="Email"
              value={email}
              onChange={(e) => setEmail(e.target.value)}
              required
            />

            <input
              aria-label="Password"
              placeholder="Password"
              type="password"
              value={password}
              onChange={(e) => setPassword(e.target.value)}
              required
            />

            <button type="submit" className="btn-primary">Sign in</button>

            <div style={{ textAlign: 'center', marginTop: 6 }}>
              <small className="auth-help">
                Don't have an account?{' '}
                <button type="button" className="btn-link" onClick={() =>
setMode('signup')}>
                  Create one
                </button>
              </small>
            </div>
          </form>
        )}

        {/* SIGNUP FORM (avatar removed, phone required) */}
        {mode === 'signup' && (
```

```jsx
          <form onSubmit={handleSignup} className="auth-form">
            {error && <div style={errorStyle}>{error}</div>}
            {success && <div style={successStyle}>{success}</div>}

            <input
              aria-label="Name"
              placeholder="Full name"
              value={name}
              onChange={(e) => setName(e.target.value)}
              required
            />

            <input
              aria-label="Email"
              placeholder="Email"
              type="email"
              value={signupEmail}
              onChange={(e) => setSignupEmail(e.target.value)}
              required
            />

            <div style={{ display: 'grid', gap: 6 }}>
              <input
                aria-label="Password"
                placeholder="Password"
                type="password"
                value={signupPassword}
                onChange={(e) => setSignupPassword(e.target.value)}
                minLength={8} // basic HTML guard
                required
              />
              {/* Live requirements message in red */}
              {!signupPasswordIsValid && signupPassword.length > 0 && (
                <small style={errorStyle}>
                  Password must be at least 8 characters and include
uppercase, lowercase, a number, and a special character.
                </small>
              )}
            </div>

            <input
              aria-label="Phone"
              placeholder="Phone"
              value={phone}
              onChange={(e) => {
```

```
                const value = e.target.value.replace(/\D/g, '') // keep
only digits
                // limit to max 10 digits
                if (value.length <= 10) {
                  setPhone(value)
                }
              }}
              required
            />
            {phone.length > 0 && phone.length < 10 && (
              <span style={{ color: 'red' }}>Phone number must be exactly
10 digits</span>
            )}
            {phone.length > 10 && (
              <span style={{ color: 'red' }}>Phone number cannot exceed 10
digits</span>
            )}

            {/* Photo upload removed */}

            <button
              type="submit"
              className="btn-primary"
              disabled={phone.length !== 10}
            >
              Create account
            </button>
          </form>
        )}
      </div>
    </div>
  )
}
```



After Login- Go to Home Page

SignupForm

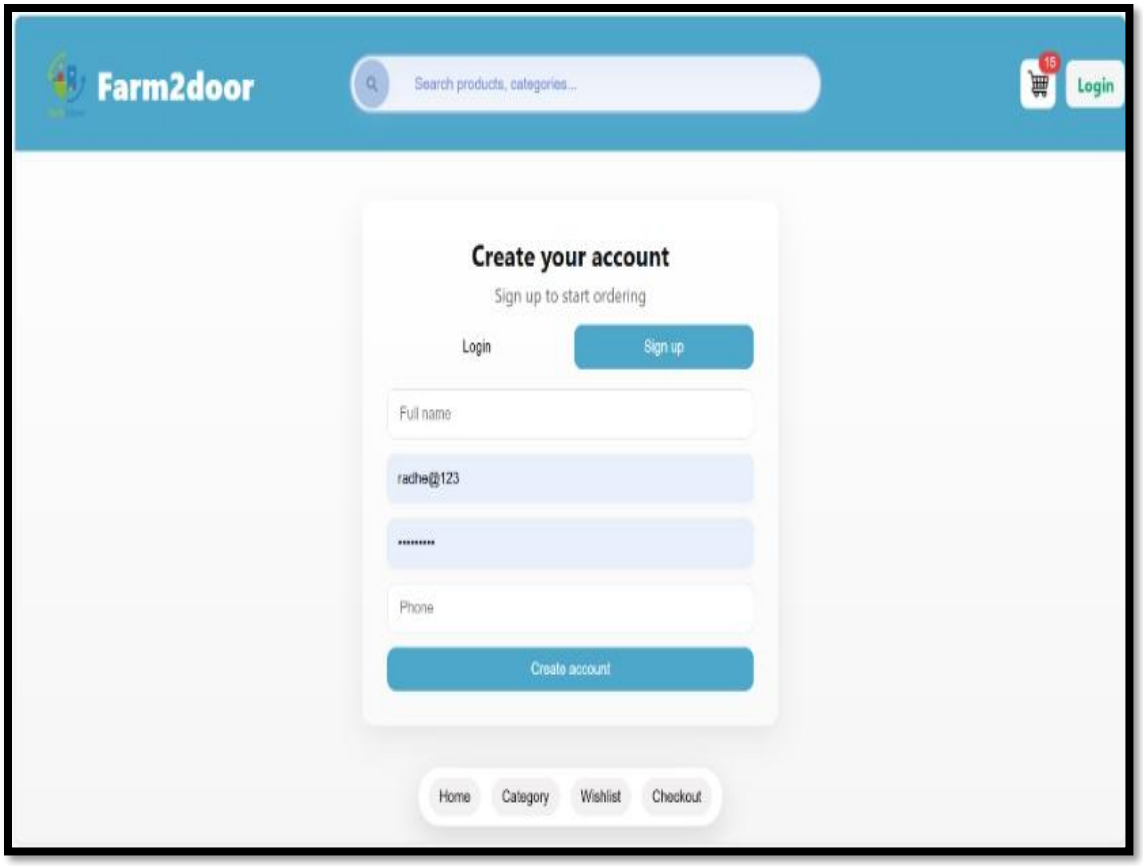
Purpose: Collects full name, email, password, phone; calls AuthContext.register.

Props/State: depends on framework; typically receives product/user objects and uses local state for form inputs.

Dependencies: AuthContext, ProductContext ,Router/Navigation, localStorage/API services.

Example Usage:

<SignupForm /> // mounted within routes; interacts with contexts and services



Sign-Up page – register a new account

```
  // SIGNUP
  function handleSignup(e) {
    e.preventDefault()
    setError(''); setSuccess('')

    if (!name || !signupEmail || !signupPassword || !phone) {
      return setError('Please fill all required fields: Name, Email,
Phone, Password.')
    }

    // enforce strong password
    if (!signupPasswordIsValid) {
      return setError(
        'Password must be at least 8 characters and include uppercase,
lowercase, a number, and a special character.'
      )
    }

    // avatar removed; pass null/omit if your AuthContext expects it
```

```
    const res = signup({ name, email: signupEmail, password:
signupPassword, phone })
    if (!res?.ok) return setError(res?.message ?? 'Signup failed')

    setSuccess('Signup successful — you are logged in')
    navigate(from, { replace: true })
  }
```

CategoryLisy & ProductCard

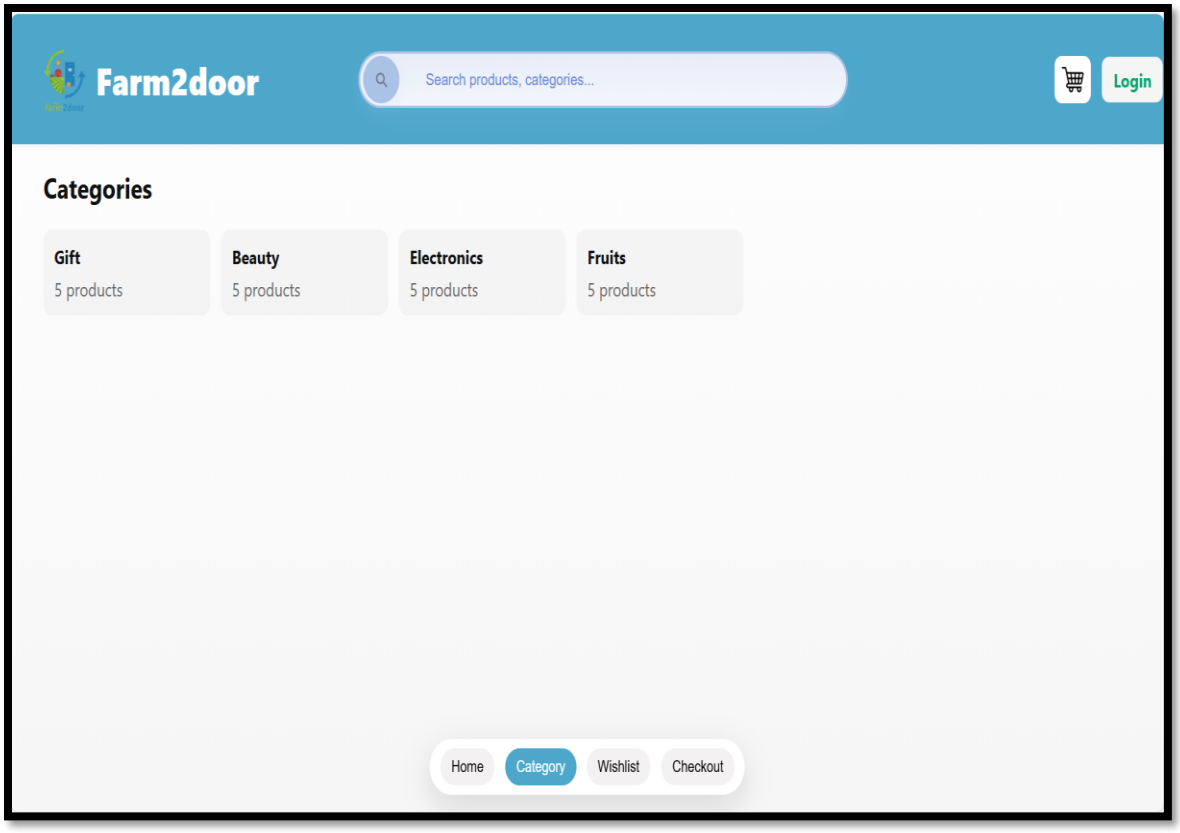
Purpose: Shows name, image, category, and price; actions: Add to Cart/Wishlist.

Props/State: depends on framework; typically receives product/user objects and uses local state for form inputs.

Dependencies: AuthContext, ProductContext ,Router/Navigation, localStorage/API services.

Example Usage:

<ProductCard /> // mounted within routes; interacts with contexts and services




```
// src/components/ProductCard.jsx
import React, { useContext, useState, useMemo } from 'react'
import { useNavigate, useLocation } from 'react-router-dom'
import { AuthContext } from '../context/AuthContext'
import { ProductContext } from '../context/ProductContext'
// import { useToast } from './context/ToastContext'
import InlineToast from './InlineToast'
import ProductModal from './ProductModal'

export default function ProductCard({
  product,
  full = false,
  useDialog = false,
  onAdd,       // optional: if parent (Home/CategorySection) wants to
handle add
  onWishlist,  // optional: if parent wants to handle wishlist
}) {
  const navigate = useNavigate()
  const location = useLocation()
```

```
  const { isLoggedIn, isAdmin } = useContext(AuthContext)
  const { addToCart, wishlist, addToWishlist, removeFromWishlist } =
    useContext(ProductContext)

  const [hover, setHover] = useState(false)
  const [open, setOpen] = useState(false)
  const [inline, setInline] = useState({ open: false, message: '', type:
'info' })

  // Safe memoized check
  const saved = useMemo(() => wishlist?.some((w) => w.id === product.id),
[wishlist, product.id])

  const handleProductClick = () => {
    if (useDialog) {
      setOpen(true)
    } else {
      navigate(`/product/${product.id}`)
    }
  }

  // Helper: redirect to login with intent
  const redirectToLoginWithIntent = (intent) => {
    navigate('/login', {
      state: {
        from: location,
        intent, // e.g. { type: 'ADD_TO_CART', product } or { type:
'TOGGLE_WISHLIST', product }
      },
      replace: false,
    })
  }

  // --- Actions ---
  const runAddToCart = () => {
    // If parent provided a handler, call it but still show a toast here
    if (typeof onAdd === 'function') {
      onAdd(product)
      // show inline popup near product
      setInline({ open: true, message: 'Added to cart', type: 'success' })
      return
    }

    addToCart(product)
    setInline({ open: true, message: 'Added to cart', type: 'success' })
```

```
  }

  const runWishlistAdd = () => {
    // If parent provided a handler, call it and still show a toast here
    if (typeof onWishlist === 'function') {
      onWishlist(product)
      setInline({ open: true, message: 'Saved to wishlist', type:
'success' })
      return
    }

    addToWishlist(product)
    setInline({ open: true, message: 'Saved to wishlist', type: 'success'
})
  }

  const runWishlistRemove = () => {
    removeFromWishlist(product.id)
    setInline({ open: true, message: 'Removed from wishlist', type: 'info'
})
  }

  return (
    <div
      onClick={handleProductClick}
      onMouseEnter={() => setHover(true)}
      onMouseLeave={() => setHover(false)}
      className={`product-card ${hover ? 'hover' : ''} ${full ? 'full' :
''}`}
      style={{ cursor: 'pointer' }}
    >
      <div className="product-media">
        <img
          src={product.image}
          alt={product.title}
          className="product-img"
          loading="lazy"
          decoding="async"
          onError={(e) => {
            e.currentTarget.onerror = null
            e.currentTarget.src = '/vite.svg'
          }}
        />
      </div>
```

```
      <div className="product-body">
        <h4 className="product-title">{product.title}</h4>
        <div className="product-category">{product.category}</div>
        <div className="product-stock" style={{ fontSize: '12px', color:
product.stock > 0 ? '#059669' : '#dc2626', marginTop: '4px', fontWeight:
500 }}>
          {product.stock > 0 ? `Stock: ${product.stock}` : 'Out of Stock'}
        </div>
      </div>

      <div className="product-footer">
        <strong className="product-
price">${Number(product.price).toFixed(2)}</strong>

        <div>
          {/* Wishlist button */}
          <button
            onClick={(e) => {
              e.stopPropagation() // prevent opening product
              if (isAdmin) {
                // block admin actions
                setInline({ open: true, message: 'Admins cannot add
products to cart', type: 'warn' });
                return;
              }

              if (!isLoggedIn) {
                // carry intent so wishlist is added right after login
                redirectToLoginWithIntent({ type: 'TOGGLE_WISHLIST',
product })
                return
              }

              if (saved) {
                runWishlistRemove()
              } else {
                runWishlistAdd()
              }
            }}
            className="wish-btn"
            aria-label={saved ? 'Remove from wishlist' : 'Add to
wishlist'}
          >
            {saved ? '♥' : '♡'}
          </button>
```

```
          {/* Add to Cart button */}
          <button
            className="add-btn"
            onClick={(e) => {
              e.stopPropagation(); // prevent opening product

              if (isAdmin) {
                // block admin actions
                setInline({ open: true, message: 'Admins cannot add
products to cart', type: 'warn' });
                return;
              }

              if (!isLoggedIn) {
                // carry intent so item is added right after login
                redirectToLoginWithIntent({ type: 'ADD_TO_CART', product
});
                return;
              }

              runAddToCart();
            }}
            aria-label="Add to cart"
            disabled={product.stock <= 0}
          >
            Add
          </button>

        </div>
      </div>

      {useDialog && open && (
        <ProductModal product={product} onClose={() => setOpen(false)} />
      )}
      <InlineToast open={inline.open} message={inline.message}
type={inline.type} onClose={() => setInline({ open: false, message: '',
type: 'info' })} />
    </div>
  )
}
```

CartPage

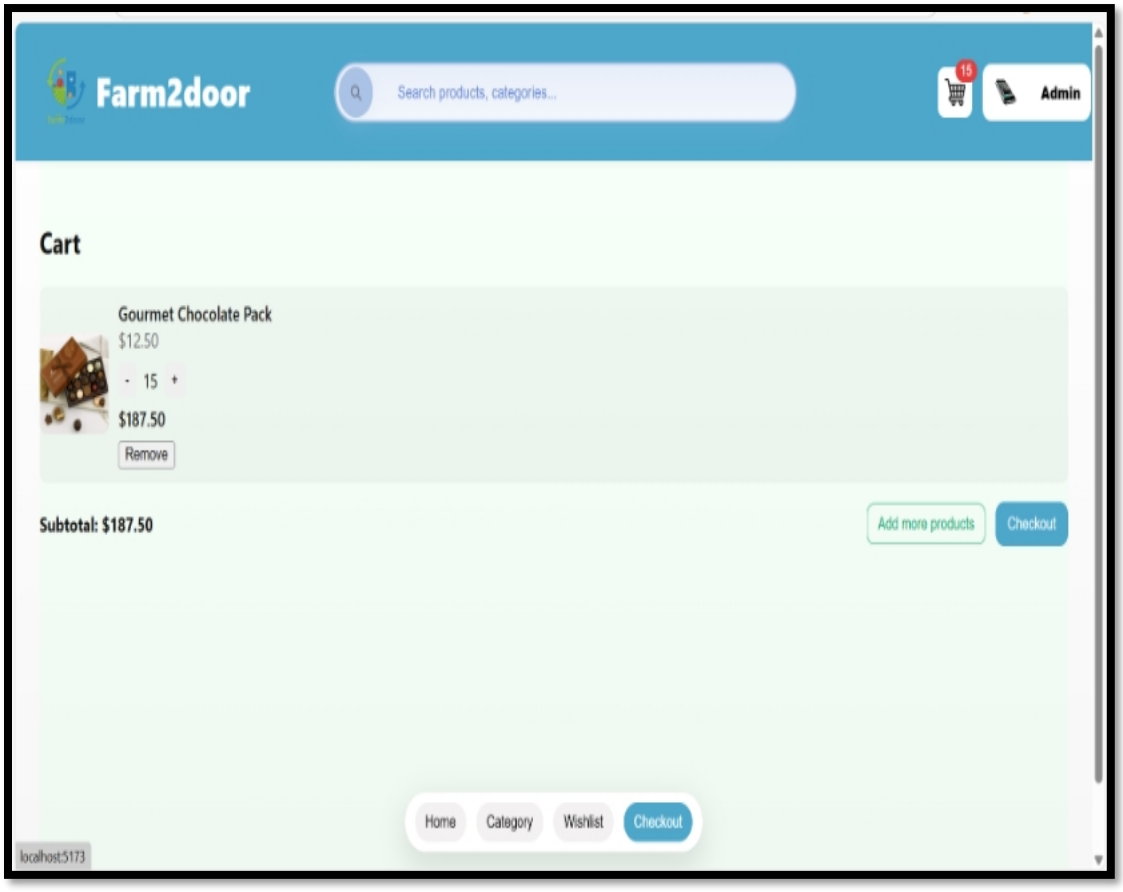
Purpose: Shows cart items, quantities, subtotal, and checkout button.

Props/State: depends on framework; typically receives product/user objects and uses local state for form inputs.

Dependencies: AuthContext, ProductContext ,Router/Navigation, localStorage/API services.

Example Usage:

<CartPage /> // mounted within routes; interacts with contexts and services



Cart page – item quantity and subtotal

```
// src/pages/Cart.jsx
import React, { useContext, useEffect } from 'react'
import { ProductContext } from '../context/ProductContext'
import { AuthContext } from '../context/AuthContext'
```

```
import { useToast } from '../context/toastcontext'
import { useNavigate, useLocation } from 'react-router-dom'

export default function Cart() {
  const { cart, removeFromCart, updateCartQty, createOrder } =
useContext(ProductContext)
  const { isLoggedIn, user } = useContext(AuthContext)
  const { showToast } = useToast()
  const navigate = useNavigate()
  const location = useLocation()

  useEffect(() => {
    if (!isLoggedIn) {
      navigate('/login', { state: { from: location } })
    }
  }, [isLoggedIn, navigate, location])

  if (!isLoggedIn) return null

  // Wait until we know if the user is logged in
  const subtotal = cart.reduce((sum, item) => {
    const price = Number(item.price) || 0
    const qty = Number(item.qty) || 1
    return sum + price * qty
  }, 0)

  if (!cart || cart.length === 0) {
    return (
      <div className="page-container cart-page">
        <h2>Cart</h2>
        <div className="empty-state" style={{ padding: '16px 0', color:
'var(--muted)' }}>
          Your cart is empty
          <div>
            <button
              type="button"
              className="btn-outline"
              onClick={() => navigate('/')}
            >
              Add more products
            </button>
          </div>
        </div>
      </div>
    )
```

```
  }

  function handleCheckout() {
    if (!cart || cart.length === 0) {
      showToast('Cart is empty', 'info')
      return
    }

    createOrder({
      items: cart.map(({ id, title, price, qty, image }) => ({
        id,
        title,
        price: Number(price) || 0,
        qty: Number(qty) || 1,
        image,
      })),
      total: subtotal,
      user: user?.username ?? null,
    })

    showToast(`Order placed for ${cart.length} item(s)`, 'success')
    navigate('/orders')
  }

  return (
    <div className="page-container cart-page">
      <h2>Cart</h2>

      <div className="cart-list">
        {cart.map((c) => {
          const price = Number(c.price) || 0
          const qty = Number(c.qty) || 1
          const lineTotal = (price * qty).toFixed(2)

          return (
            <div key={c.id} className="cart-item" style={{ display:
'flex', gap: 12, alignItems: 'center', padding: '10px 0', borderBottom:
'1px solid var(--border, #e5e7eb)' }}>
              <img
                src={c.image}
                alt={c.title}
                style={{
                  width: 80,
                  height: 80,
                  objectFit: 'cover',
```

```
                  borderRadius: 8,
                  flexShrink: 0,
                }}
                onError={(e) => {
                  e.currentTarget.onerror = null
                  e.currentTarget.src = '/vite.svg'
                }}
              />

              <div className="cart-item-info" style={{ flex: 1, minWidth:
0 }}>
                <div className="cart-item-title" style={{ fontWeight: 600
}}>{c.title}</div>
                <div className="cart-item-price" style={{ color: 'var(--
muted)' }}>${price.toFixed(2)}</div>

                <div className="qty-controls" style={{ display: 'inline-
flex', gap: 8, alignItems: 'center', marginTop: 8 }}>
                  <button
                    type="button"
                    onClick={() => {
                      const nextQty = qty - 1
                      if (nextQty <= 0) {
                        removeFromCart(c.id)
                        showToast('Item removed from cart', 'info')
                      } else {
                        updateCartQty(c.id, nextQty)
                        showToast(`Quantity updated to ${nextQty}`,
'success')
                      }
                    }}
                  >
                    -
                  </button>

                  <span className="qty">{qty}</span>

                  <button
                    type="button"
                    onClick={() => {
                      updateCartQty(c.id, qty + 1)
                      showToast(`Quantity updated to ${qty + 1}`,
'success')
                    }}
                  >
```

```
                  +
                </button>
              </div>

              <div className="line-total" style={{ fontWeight: 600,
marginTop: 6 }}>
                ${lineTotal}
              </div>

              <button
                type="button"
                className="btn-danger"
                style={{ marginTop: 8 }}
                onClick={() => {
                  removeFromCart(c.id)
                  showToast('Item removed from cart and stock restored',
'info')
                }}
              >
                Remove
              </button>
            </div>
          </div>
        )
      })}
    </div>

    <div className="cart-footer" style={{ display: 'flex',
justifyContent: 'space-between', alignItems: 'center', marginTop: 16 }}>
      <div className="subtotal" style={{ fontWeight: 700 }}>
        Subtotal: ${subtotal.toFixed(2)}
      </div>
      <div className="cart-actions">
        <button type="button" className="btn-outline" onClick={() =>
navigate('/')}>
          Add more products
        </button>
        <button type="button" className="btn-primary"
onClick={handleCheckout}>
          Checkout
        </button>
      </div>
    </div>
  </div>
)
```

```
}
```

WishlistPage

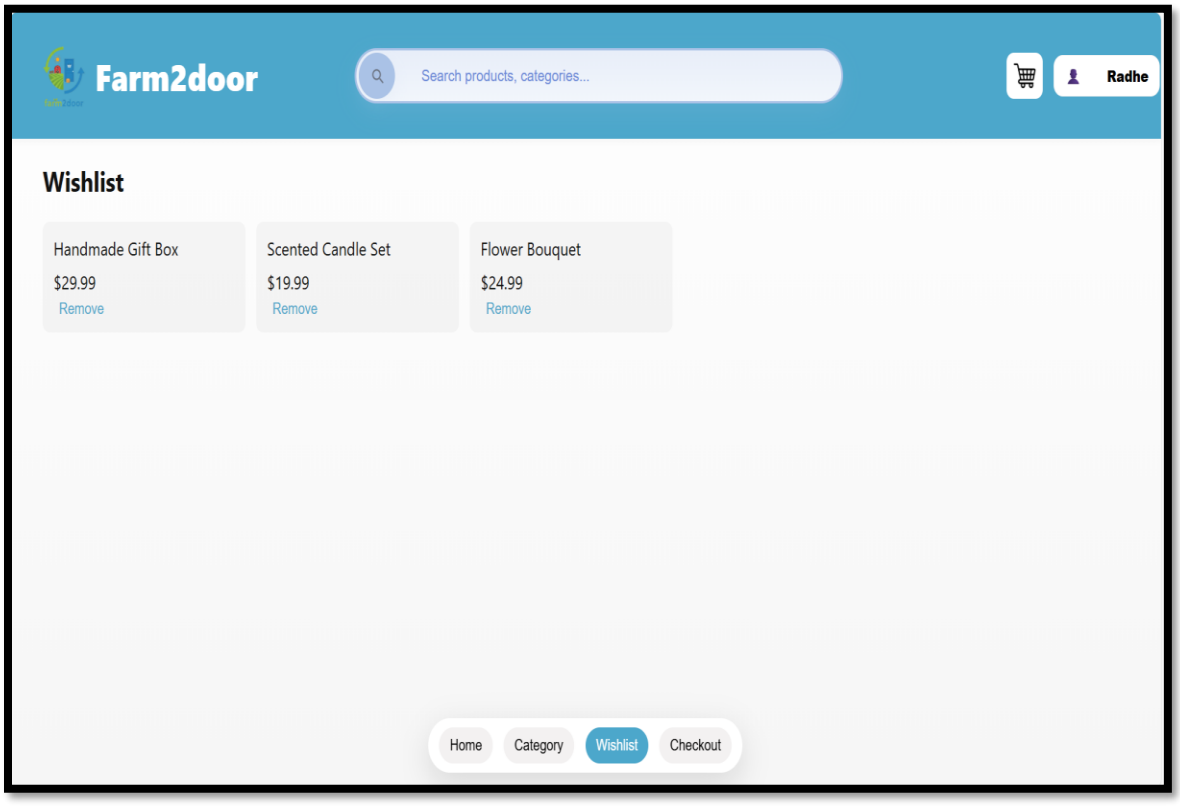
Purpose: Shows saved products; allows remove or move to cart.

Props/State: depends on framework; typically receives product/user objects and uses local state for form inputs.

Dependencies: AuthContext, ProductContext ,Router/Navigation, localStorage/API services.

Example Usage:

<WishlistPage /> // mounted within routes; interacts with contexts and services



```jsx
import React, { useContext } from 'react';
import { ProductContext } from '../context/ProductContext';
import { AuthContext } from '../context/AuthContext';
import { useNavigate } from 'react-router-dom';
import { useToast } from '../context/toastcontext';
import InlineToast from '../components/InlineToast'

export default function Wishlist() {
  const { wishlist, removeFromWishlist } = useContext(ProductContext);
  const { isLoggedIn } = useContext(AuthContext);
  const nav = useNavigate();
  const { showToast } = useToast();
  const [inline, setInline] = React.useState({ open: false, message: '',
type: 'info', id: null })

  if (!isLoggedIn) {
    return (
      <div style={{ padding: 12 }}>
        <h3>Please login to view your wishlist</h3>
        <button onClick={() => nav('/login')}>Go to Login</button>
      </div>
    );
  }

  return (
    <div className="page-container">
      <h2>Wishlist</h2>
      <div className="wishlist-grid">
        {wishlist.length === 0 && <div>No items saved.</div>}
        {wishlist.map((p) => (
          <div key={p.id} className="wish-card">
            <div className="wish-title">{p.title}</div>
            <div className="wish-price">${p.price}</div>
            <div>
              <button className="btn-link" onClick={() => {
removeFromWishlist(p.id); setInline({ open: true, message: 'Removed from
wishlist', type: 'info', id: p.id }); }}>
                Remove
              </button>
              {inline.open && inline.id === p.id && (
                <InlineToast open={inline.open} message={inline.message}
type={inline.type} onClose={() => setInline({ open: false, message: '',
type: 'info', id: null })} />
              )}
            </div>
```

```
          </div>
        ))}
      </div>
    </div>
  );
}
```

ProfilePanel

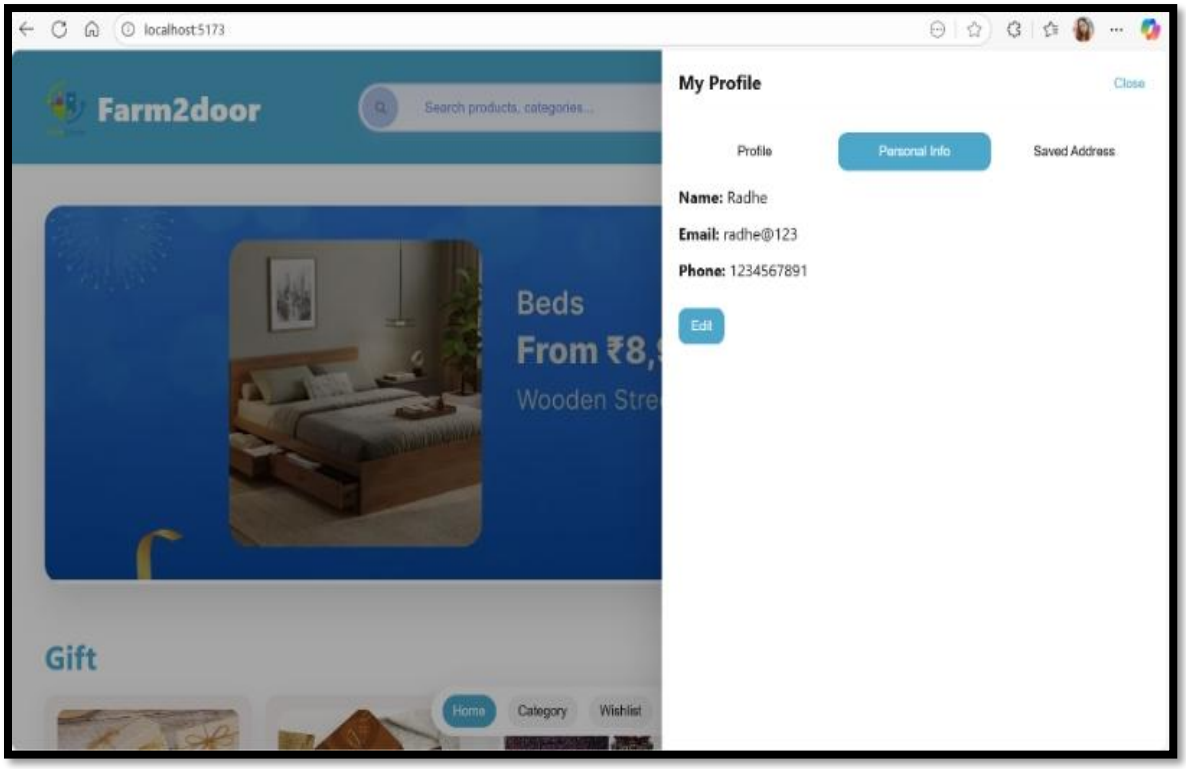
Purpose: Displays personal info and saved addresses; edit/add dialogs.

Props/State: depends on framework; typically receives product/user objects and uses local state for form inputs.

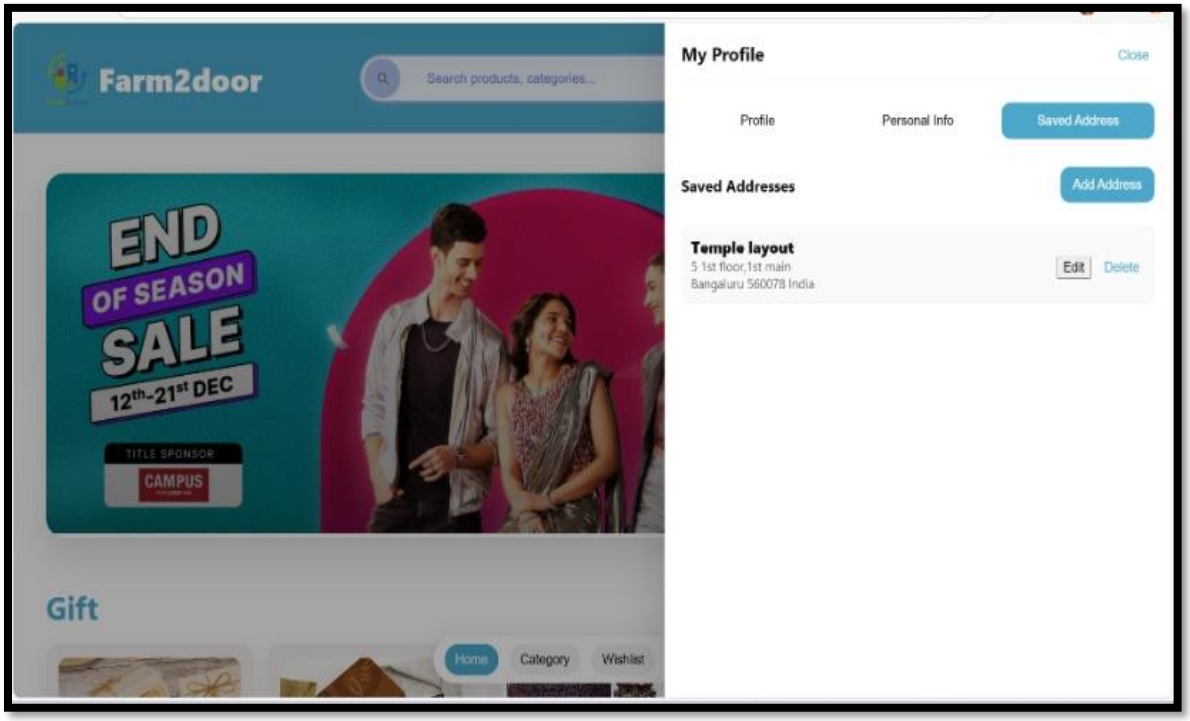
Dependencies: AuthContext, ProductContext ,Router/Navigation, localStorage/API services.

Example Usage:

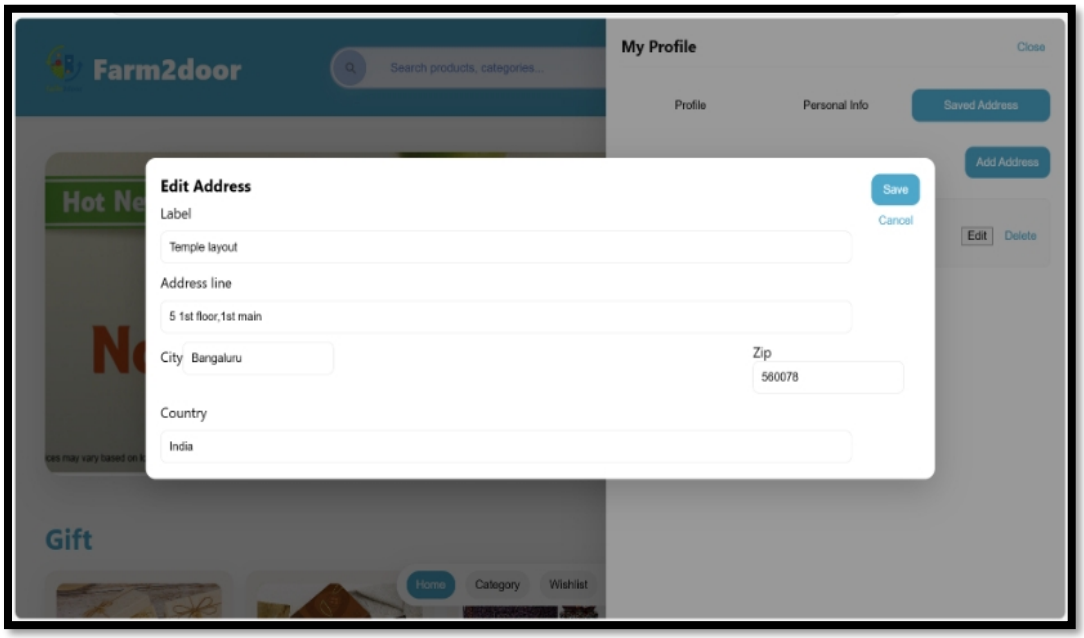
<ProfilePanel /> // mounted within routes; interacts with contexts and services.



Profile – personal info view & edit form



Profile – saved addresses list & edit address form



Profile – add address form

```jsx
import React, { useContext, useState, useEffect, useRef } from 'react';
import { useToast } from '../context/toastcontext';
import { AuthContext } from '../context/AuthContext';
import ConfirmModal from '../components/ConfirmModal';
import AddressForm from '../components/AddressForm';

export default function Profile() {
  const { user, updateProfile, addAddress, editAddress, removeAddress } =
useContext(AuthContext);
  const [view, setView] = useState('profile'); // 'profile' | 'personal' |
'addresses'
  const [editingPersonal, setEditingPersonal] = useState(false);
  const [drawerOpen, setDrawerOpen] = useState(false);
  const [name, setName] = useState(user?.name || '');
  const [phone, setPhone] = useState(user?.phone || '');
  const [avatarPreview, setAvatarPreview] = useState(user?.avatar ||
null);
  const [addressModal, setAddressModal] = useState({ open: false, initial:
null });
  const [confirm, setConfirm] = useState({ open: false });
  const fileRef = useRef(null);
  const drawerRef = useRef(null);
  const closeBtnRef = useRef(null);

  // Sync local state when user context changes
  useEffect(() => {
    setName(user?.name || '');
    setPhone(user?.phone || '');
    setAvatarPreview(user?.avatar || null);
  }, [user]);

  // Listen for global open events so other components can open the drawer
  useEffect(() => {
    function handleOpen(e) {
      const v = e?.detail?.view || 'profile';
      setView(v);
      setDrawerOpen(true);
    }
    window.addEventListener('farm2door:openProfile', handleOpen);
    return () => window.removeEventListener('farm2door:openProfile',
handleOpen);
  }, []);

  // When drawer opens, trap focus to the drawer and prevent background
scroll.
```

```
  useEffect(() => {
    function onKey(e) {
      if (e.key === 'Escape') setDrawerOpen(false);
    }

    if (drawerOpen) {
      document.body.style.overflow = 'hidden';
      window.addEventListener('keydown', onKey);
      // focus first tab button inside drawer
      setTimeout(() => {
        try {
          const btn = drawerRef.current?.querySelector('.nav-btn');
          if (btn) btn.focus();
          else closeBtnRef.current?.focus();
        } catch { }
      }, 50);
    }

    return () => {
      document.body.style.overflow = '';
      window.removeEventListener('keydown', onKey);
    };
  }, [drawerOpen]);

  function toBase64(file) {
    return new Promise((res, rej) => {
      const fr = new FileReader();
      fr.onload = () => res(fr.result);
      fr.onerror = rej;
      fr.readAsDataURL(file);
    });
  }

  async function handleAvatarInput(e) {
    const file = e.target.files?.[0] || null;
    if (!file) return;
    try {
      const b64 = await toBase64(file);
      // Update immediately in UI and context
      setAvatarPreview(b64);
      updateProfile({ avatar: b64 });
      if (fileRef.current) fileRef.current.value = ''; // reset the file
input so that user can re-select the same file again
    } catch (err) {
      console.error(err);
```

```
      const { showToast } = useToast();
      showToast('Failed to read image', 'error')
    }
  }

  // Do not render any UI until the drawer is opened. The component
remains mounted
  // so it can listen for `farm2door:openProfile` events.
  if (!drawerOpen) return null;

  function openAdd() {
    setAddressModal({ open: true, initial: { label: '', line: '', city:
'', zip: '', country: '' } });
  }

  function openEditAddress(a) {
    setAddressModal({ open: true, initial: { ...a } });
  }

  function handleSaveAddress(addr) {
    // If addr has an id, treat as edit; otherwise add (AuthContext will
create id)
    if (!addr.line || addr.line.trim().length < 5) {
      showToast('Please enter a valid address', 'warn');
      return;
    }

    if (!addr.city || addr.city.trim().length < 2) {
      showToast('Please enter a valid city', 'warn');
      return;
    }

    if (!/^\d{6}$/.test(addr.zip)) {
      showToast('PIN code must be 6 digits', 'warn');
      return;
    }

    if (!addr.country || addr.country.trim().length < 2) {
      showToast('Please enter a valid country', 'warn');
      return;
    }

    if (addr?.id) editAddress(addr.id, addr);
    else addAddress(addr);
  }
```

```
  function confirmDeleteAddress(id) {
    setConfirm({ open: true, title: 'Delete address', message: 'Delete
this address?', onConfirm: () => { removeAddress(id); setConfirm({ open:
false }); } });
  }

  function savePersonal() {
    // simple validation
    const { showToast } = useToast();
    if (!name || name.trim().length < 2) {
      alert('Please enter a valid name', 'warn')
      return
    }
    if (phone && !/^[0-9+\- ()]{6,10}$/.test(phone)) {
      showToast('Please enter a valid phone number', 'warn')
      return
    }
    updateProfile({ name: name.trim(), phone: phone.trim() });
    setEditingPersonal(false);
  }
  return (
    <div className="page-container">
      {/* Drawer implementation: backdrop + drawer panel (drawer is opened
via app events) */}
      {drawerOpen && (
        <div className="drawer-backdrop" onClick={() =>
setDrawerOpen(false)}>
          <div className="drawer" role="dialog" onClick={(e) =>
e.stopPropagation()}>
            <div className="drawer-header">
              <h3 style={{ margin: 0 }}>My Profile</h3>
              <div>
                <button className="btn-link" onClick={() =>
setDrawerOpen(false)}>Close</button>
              </div>
            </div>


            <div className="auth-tabs" role="tablist" style={{ display:
'flex', gap: 8, marginTop: 12 }}>
              <button className={`nav-btn ${view === 'profile' ? 'active'
: ''}`} onClick={() => setView('profile')}>Profile</button>
              <button className={`nav-btn ${view === 'personal' ? 'active'
: ''}`} onClick={() => setView('personal')}>Personal Info</button>
```

```
            <button className={`nav-btn ${view === 'addresses' ?
'active' : ''}`} onClick={() => setView('addresses')}>Saved
Address</button>
          </div>

          <div className="drawer-body">
            {!user ? (
              <div style={{ padding: 18 }}>
                <h3>Please login</h3>
                <p className="muted">You must be logged in to view
profile details.</p>
                <div style={{ marginTop: 12 }}>
                  <button className="btn-primary" onClick={() => {
setDrawerOpen(false); window.location.href = '/login'; }}>Go to
Login</button>
                </div>
              </div>
            ) : (
              <>
                {view === 'profile' && (
                  <div style={{ display: 'flex', gap: 16, alignItems:
'center' }}>
                    {avatarPreview ? <img src={avatarPreview}
alt="avatar" className="avatar-large" /> : <div className="avatar-large
avatar-empty">No avatar</div>}
                    <div>
                      <div style={{ fontSize: 18, fontWeight: 700
}}>{user.name}</div>
                      <div style={{ marginTop: 8 }}>
                        <button className="btn-primary" onClick={() =>
fileRef.current && fileRef.current.click()}>Change Avatar</button>
                      </div>
                    </div>
                  </div>
                )}

                {view === 'personal' && (
                  <div>
                    {!editingPersonal ? (
                      <div className="profile-info">
                        <div><strong>Name:</strong> {user.name}</div>
                        <div><strong>Email:</strong> {user.email}</div>
                        <div><strong>Phone:</strong> {user.phone}</div>
                        <div className="profile-actions" style={{
marginTop: 12 }}>
```

```
                          <button onClick={() =>
setEditingPersonal(true)} className="btn-primary">Edit</button>
                        </div>
                      </div>
                    ) : (
                      <div className="profile-edit">
                        <label>Full name</label>
                        <input
                          type="text"
                          value={name}
                          onChange={(e) => setName(e.target.value)}
                        />
                        <label>Phone</label>
                        <input
                          value={phone}
                          maxLength={10}
                          inputMode="numeric"
                          onChange={(e) =>
setPhone(e.target.value.replace(/\D/g, ''))}
                        />

                        <div className="profile-actions">
                          <button onClick={savePersonal} className="btn-
primary">Save</button>
                          <button onClick={() => {
setEditingPersonal(false); setName(user?.name || ''); setPhone(user?.phone
|| ''); }} className="btn-link">Cancel</button>
                        </div>
                      </div>
                    )}
                  </div>
                )}

                {view === 'addresses' && (
                  <div>
                    <div style={{ display: 'flex', justifyContent:
'space-between', alignItems: 'center' }}>
                      <h4>Saved Addresses</h4>
                      <div>
                        <button className="btn-primary"
onClick={openAdd}>Add Address</button>
                      </div>
                    </div>
```

```
                    {(user.addresses || []).length === 0 && <div
style={{ marginTop: 12 }}>No addresses yet.</div>}

                    {(user.addresses || []).map((a) => (
                      <div key={a.id} className="address-card">
                        <div>
                          <div className="address-
label"><strong>{a.label || 'Address'}</strong></div>
                          <div className="address-line">{a.line}</div>
                          <div className="address-line">{a.city} {a.zip}
{a.country}</div>
                        </div>
                        <div className="address-actions">
                          <button onClick={() =>
openEditAddress(a)}>Edit</button>
                          <button onClick={() =>
confirmDeleteAddress(a.id)} className="btn-link">Delete</button>
                        </div>
                      </div>
                    ))}
                  </div>
                )}
              </>
            )}
          </div>

          {/* hidden file input for avatar changes inside drawer */}
          <input ref={fileRef} type="file" accept="image/*" style={{
display: 'none' }} onChange={handleAvatarInput} />

          <AddressForm open={addressModal.open}
initial={addressModal.initial} onSave={handleSaveAddress} onCancel={() =>
setAddressModal({ open: false, initial: null })} />
          <ConfirmModal open={confirm.open} title={confirm.title}
message={confirm.message} onCancel={() => setConfirm({ open: false })}
onConfirm={() => { if (confirm.onConfirm) confirm.onConfirm(); else
setConfirm({ open: false }); }} />
        </div>
      </div>
    )}
  </div>
);
}
```

Checkout

Purpose:

Displays all items added to the cart, along with their name, price, quantity, and subtotal. Provides options to update quantities, remove items, add more products, and proceed to checkout for order confirmation.

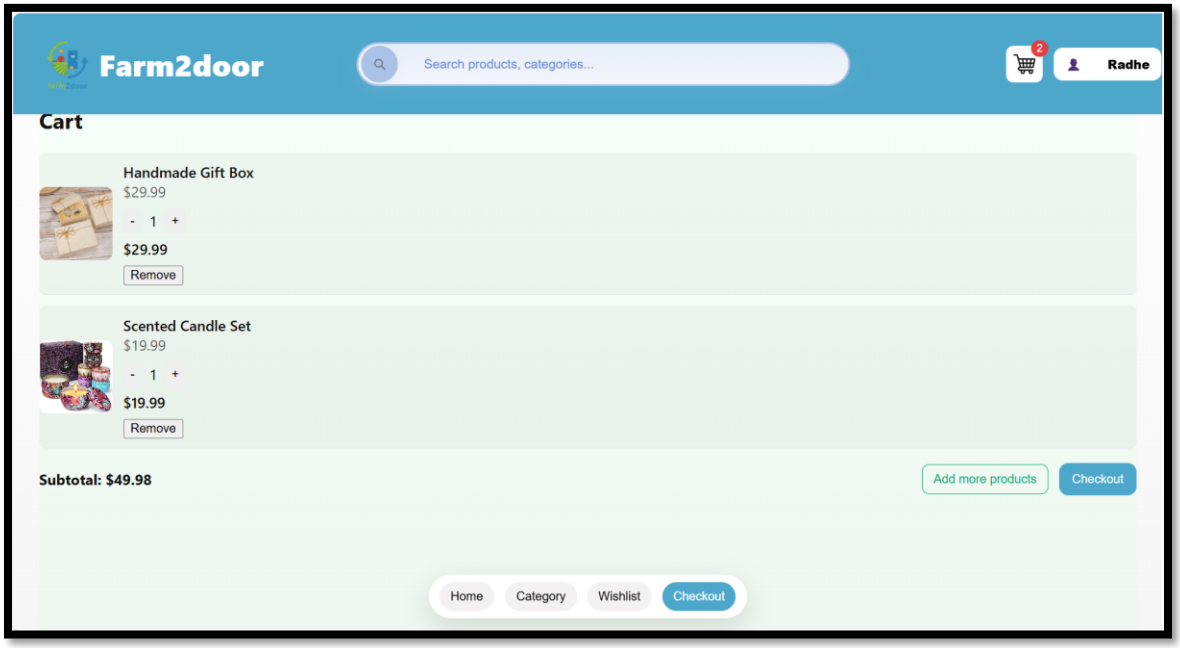
Props/State:

Depends on framework; typically receives cart items as props and uses local state for quantity updates and subtotal calculation.

Dependencies:

CartContext, ProductContext, AuthContext (for logged-in checks), Router/Navigation, localStorage/API services.

<Checkout /> // Mounted within protected route; interacts with CartContext and AuthContext




```
function handleCheckout() {
  if (!cart || cart.length === 0) {
    showToast('Cart is empty', 'info')
    return
  }
```

```
  createOrder({
    items: cart.map(({ id, title, price, qty, image }) => ({
      id,
      title,
      price: Number(price) || 0,
      qty: Number(qty) || 1,
      image,
    })),
    total: subtotal,
    user: user?.username ?? null,
  })

  showToast(`Order placed for ${cart.length} item(s)`, 'success')
  navigate('/orders')
}
```

OrderHistory

Purpose:

Displays a list of all past orders for the logged-in user. Each entry includes order ID, date/time, product details, quantity, and total amount. Helps users track previous purchases and manage returns or reorders.

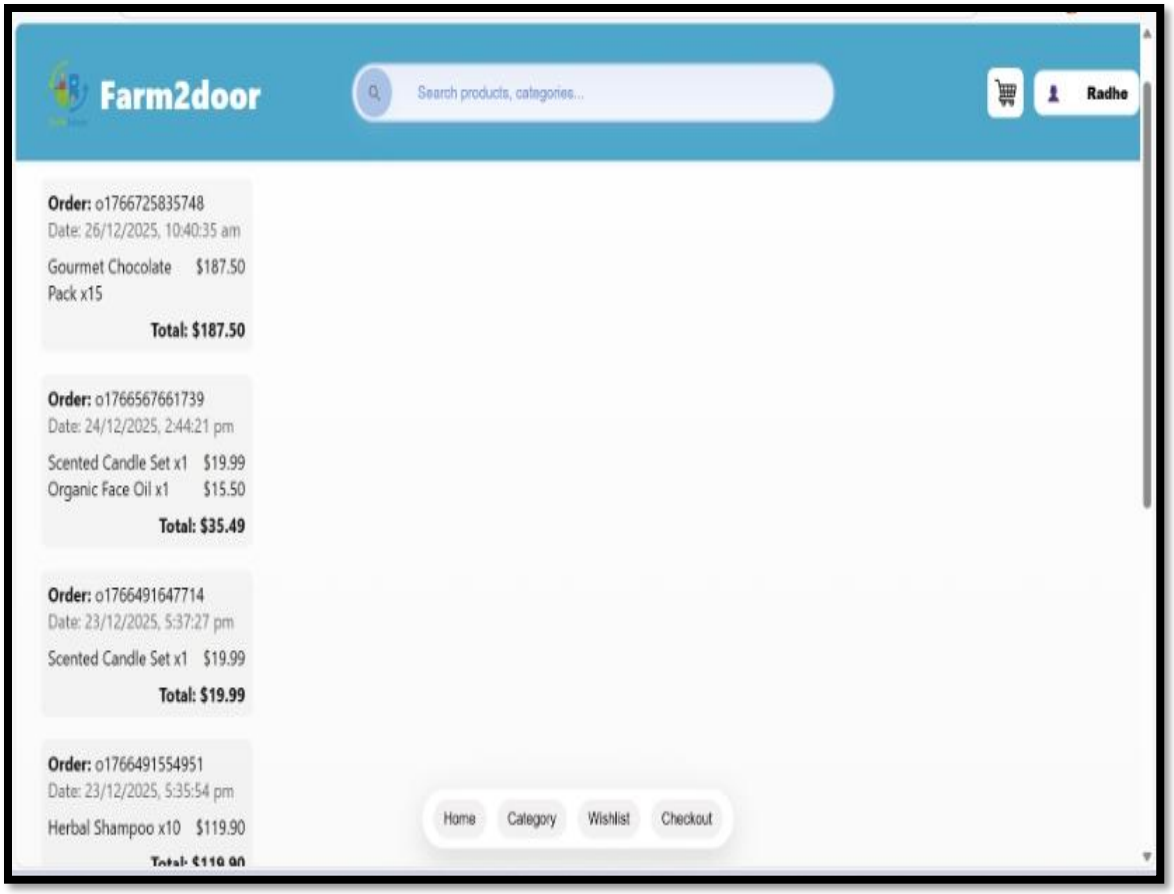
Props/State:

Receives user order data from context or API; uses local state for rendering and pagination if needed.

Dependencies:

AuthContext (to identify user), OrderContext or API service, Router/Navigation.

<OrderHistory /> // Mounted within user dashboard; fetches and displays order records



Orders page – past orders summary

```
import React, { useContext } from 'react';
import { ProductContext } from '../context/ProductContext';
import { AuthContext } from '../context/AuthContext';
import { useNavigate } from 'react-router-dom';

export default function OrderHistory() {
  const { orders } = useContext(ProductContext);
  const { isLoggedIn } = useContext(AuthContext);
  const nav = useNavigate();

  if (!isLoggedIn) {
    return (
      <div className="page-container">
        <h3>Please login to view orders</h3>
        <button onClick={() => nav('/login')}>Go to Login</button>
      </div>
    );
  }
```

```
  if (!orders || orders.length === 0) return (<div className="page-
container"><h3>No orders yet</h3></div>);

  return (
    <div className="page-container">
      <h2>Order History</h2>
      <div style={{ display: 'flex', flexDirection: 'column', gap: 18 }}>
        {orders.map((o) => (
          <div key={o.id} className="admin-card">
            <div><strong>Order:</strong> {o.id}</div>
            <div style={{ color: 'var(--muted)' }}>Date: {new
Date(o.date).toLocaleString()}</div>
            <div style={{ marginTop: 8 }}>
              {o.items.map((it) => (
                <div key={it.id} style={{ display: 'flex', justifyContent:
'space-between' }}>
                  <div>{it.title} x{it.qty}</div>
                  <div>${(it.price * it.qty).toFixed(2)}</div>
                </div>
              ))}
            </div>
            <div style={{ marginTop: 8, textAlign: 'right'
}}><strong>Total: ${o.total.toFixed(2)}</strong></div>
          </div>
        ))}
      </div>
    </div>
  );
}
```

AdminLogin:

Purpose:

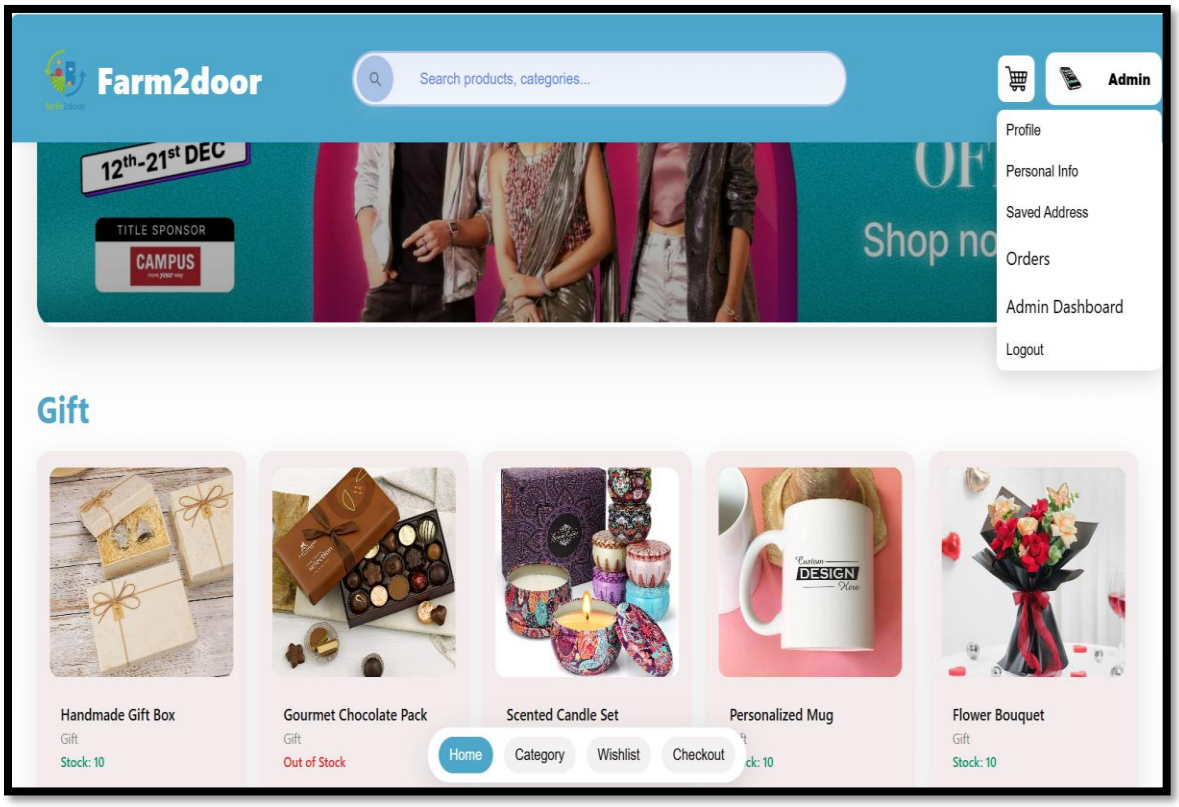
Provides a secure interface for administrators to log in and access the Admin Dashboard. It validates credentials and ensures only authorized users can perform product management and order-related tasks.

Props/State:

Receives admin credentials (email and password) as input; uses local state for form handling and error messages.

Dependencies:
AuthContext (for authentication and role verification), Router/Navigation (for redirecting to Admin Dashboard), localStorage/API services for token persistence.

<AdminLogin /> // Mounted within protected route; verifies admin role and redirects to dashboard



AdminDashboard

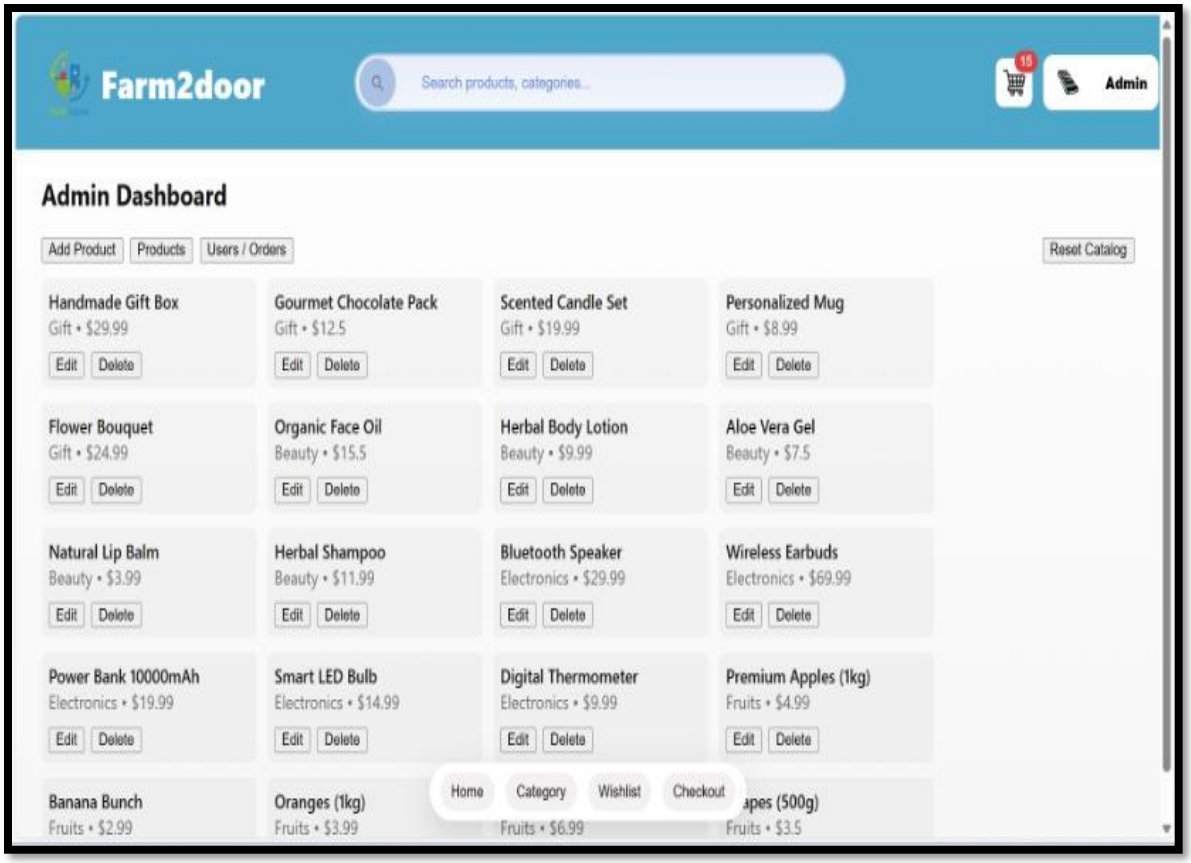
Purpose: Lists products; actions: Edit/Delete; navigation to Add Product and Users/Orders.

Props/State: depends on framework; typically receives product/user objects and uses local state for form inputs.

Dependencies: AuthContext, ProductContext ,Router/Navigation, localStorage/API services.

Example Usage:

<AdminDashboard /> // mounted within routes; interacts with contexts and services.



Admin Dashboard – product cards with Edit/Delete

AdminProductForm

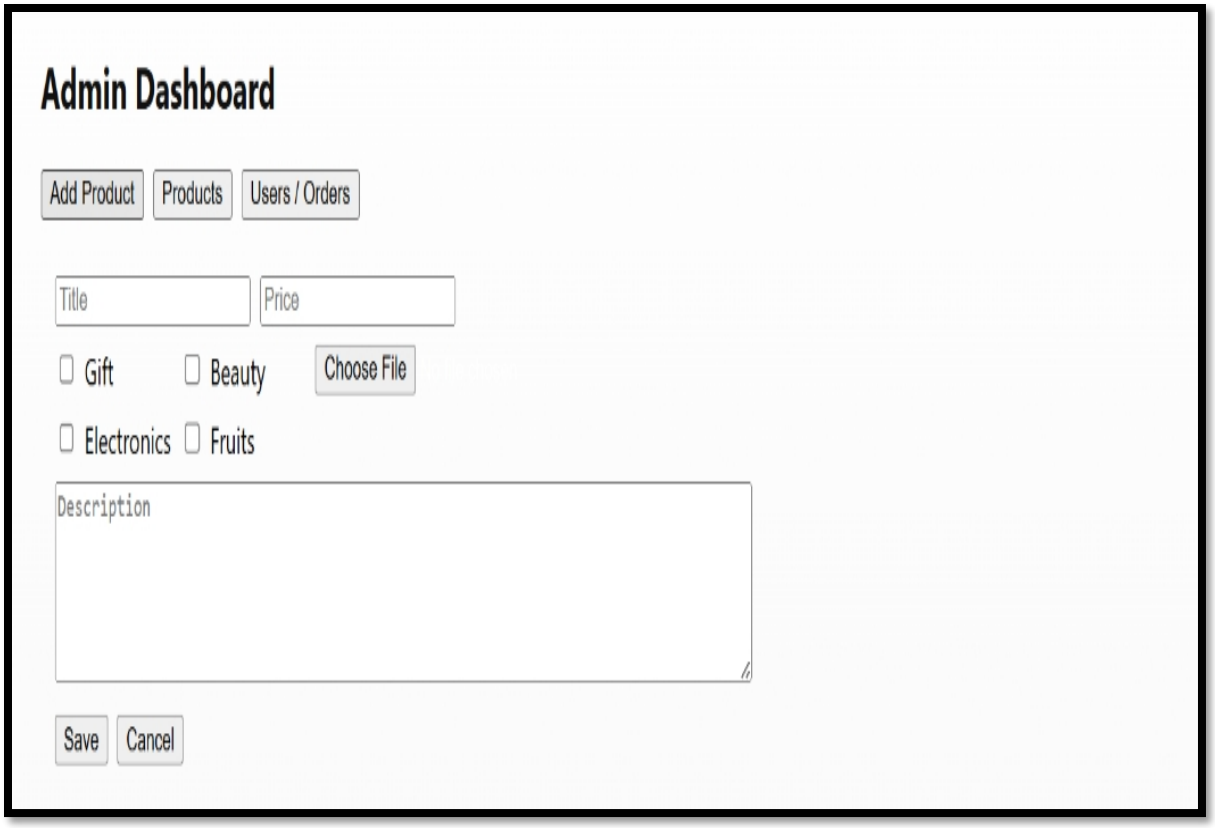
Purpose: Inputs: title, price, category, image, description; actions: Save/Cancel.

Props/State: depends on framework; typically receives product/user objects and uses local state for form inputs.

Dependencies: AuthContext, ProductContext ,Router/Navigation, localStorage/API services.

Example Usage:

<AdminProductForm /> // mounted within routes; interacts with contexts and services.



Admin Dashboard – add product form

```
//AdminDashboard.jsx
import React, { useContext, useState } from 'react';
import { ProductContext } from '../context/ProductContext';
import { AuthContext } from '../context/AuthContext';
import ProductForm from './ProductForm';
import ConfirmModal from '../components/ConfirmModal';

export default function AdminDashboard() {
  const { products, adminAddProduct, adminEditProduct, adminDeleteProduct,
resetCatalog, orders } = useContext(ProductContext);
  const { isAdmin, getAllUsers, adminDeleteUser } =
useContext(AuthContext);
  const [editing, setEditing] = useState(null);
  const [view, setView] = useState('products'); // 'products' | 'users'
  const [confirm, setConfirm] = useState({ open: false });

  if (!isAdmin) return <div className="page-container">Access denied.
Admins only.</div>;

  const users = getAllUsers();
```

```
  function confirmAction(opts) {
    setConfirm({ open: true, ...opts });
  }

  return (
    <div className="page-container">
      <h2>Admin Dashboard</h2>

      <div className="admin-controls">
        <div className="admin-controls-left">
          <button onClick={() => setEditing({})}>Add Product</button>
          <button onClick={() => setView('products')}>Products</button>
          <button onClick={() => setView('users')}>Users / Orders</button>
        </div>
        <div className="admin-controls-right">
          <button onClick={() => confirmAction({ title: 'Reset catalog',
message: 'Restore sample products? This will replace the catalog with the
original sample products.', onConfirm: () => { resetCatalog();
setConfirm({ open: false }); } })}>Reset Catalog</button>
        </div>
      </div>

      {editing && <ProductForm initial={editing} onCancel={() =>
setEditing(null)} onSave={(formData) => {
        // Convert FormData to plain object
        const productData = {
          title: formData.get('title'),
          price: Number(formData.get('price')),
          description: formData.get('description'),
          category: formData.getAll('category[]'), // gets all category
values
          image: editing.image || '', // keep existing image if not
changed
          stock: editing.stock || 10, // default stock for new products
        };

        // If editing (has id), add it
        if (editing.id) {
          productData.id = editing.id;
          adminEditProduct(productData);
        } else {
          adminAddProduct(productData);
        }
        setEditing(null);
```

```
      }} />}

      {view === 'products' && (
        <div className="admin-grid">
          {products.map((p) => (
            <div key={p.id} className="admin-card">
              <div className="admin-title">{p.title}</div>
              <div className="admin-meta">{p.category} • ${p.price}</div>
              <div className="admin-actions">
                <button onClick={() => setEditing(p)}>Edit</button>
                <button className="btn-spaced" onClick={() =>
confirmAction({ title: 'Delete product', message: `Delete
\"${p.title}\"?`, onConfirm: () => { adminDeleteProduct(p.id);
setConfirm({ open: false }); } })}>Delete</button>
              </div>
            </div>
          ))}
        </div>
      )}

      {view === 'users' && (
        <div className="admin-users-column">
          <h3>Users</h3>
          <div className="admin-grid">
            {users.map((u) => (
              <div key={u.id} className="admin-card">
                <div className="admin-title">{u.name || u.email}</div>
                <div className="admin-meta">{u.email} • {u.role}</div>
                <div className="admin-actions">
                  <button onClick={() => confirmAction({ title: 'Delete
user', message: `Delete user ${u.email}?`, onConfirm: () => {
adminDeleteUser(u.id); setConfirm({ open: false }); } })}>Delete</button>
                </div>
              </div>
            ))}
          </div>

          <h3 className="admin-orders-title">Orders</h3>
          <div className="admin-grid">
            {orders.map((o) => (
              <div key={o.id} className="admin-card">
                <div className="admin-title">Order {o.id}</div>
                <div className="admin-meta">User: {o.user?.email ||
'guest'} • {new Date(o.date).toLocaleString()}</div>
                <div className="admin-order-meta">
```

```
                  <div className="admin-order-items">{o.items?.length ||
0} items • ${o.total}</div>
                </div>
              </div>
            ))}
          </div>
        </div>
      )}

      <ConfirmModal
        open={confirm.open}
        title={confirm.title}
        message={confirm.message}
        onCancel={() => setConfirm({ open: false })}
        onConfirm={() => { if (confirm.onConfirm) confirm.onConfirm();
else setConfirm({ open: false }); }}
      />
    </div>
  );
}
```

7. Contexts / APIs

Global Contexts:

AuthContext – holds current user, roles, and auth tokens; functions: login(email, password), logout(), register(payload).

ProductContext– stores products, filters, and operations: addProduct(item), updateProduct(id, changes), deleteProduct(id), resetCatalog().

CartContext – manage cart items: addToCart(id, qty), updateQty(id, qty), removeFromCart(id), clearCart().

WishlistContext – manage wishlist: addToWishlist(id), removeFromWishlist(id).

Example (pseudo-JS):
AuthContext.login = async (email, password) => { /* verify, set token, role */ };
ProductContext.addProduct = (p) => setProducts([...products, p]);
WishlistContext.addToWishlist = (id) => setWishlist([...wishlist, id]);

Data Persistence: localStorage for demo; can be replaced by REST/GraphQL backend. Tokens/carts/wishlist saved per user session.

```jsx
//AuthContext.jsx
import React, { createContext, useEffect, useState } from 'react';

export const AuthContext = createContext();

const USERS_KEY = 'farm2door_users';
const CURRENT_USER_KEY = 'farm2door_user';

function loadUsers() {
  try {
    const raw = localStorage.getItem(USERS_KEY);
    if (!raw) {
      // create a default admin user
      const admin = [{ id: 'u_admin', name: 'Admin', email:
'admin@farm2door.local', password: 'admin', role: 'admin', addresses: [],
avatar: null, phone: '' }];
      localStorage.setItem(USERS_KEY, JSON.stringify(admin));
      return admin;
    }
    return JSON.parse(raw);
  } catch {
    return [];
  }
}

function saveUsers(users) {
  try {
    localStorage.setItem(USERS_KEY, JSON.stringify(users));
  } catch {}
}

export function AuthProvider({ children }) {
  const [users, setUsers] = useState(() => loadUsers());

  const [user, setUser] = useState(() => {
    try {
      const raw = localStorage.getItem(CURRENT_USER_KEY);
      return raw ? JSON.parse(raw) : null;
    } catch {
      return null;
    }
  });

  useEffect(() => saveUsers(users), [users]);
```

```
  useEffect(() => {
    if (user) localStorage.setItem(CURRENT_USER_KEY,
JSON.stringify(user));
    else localStorage.removeItem(CURRENT_USER_KEY);
  }, [user]);

  function findUserByEmail(email) {
    return users.find((u) => u.email.toLowerCase() === (email ||
'').toLowerCase());
  }

  function signup({ name, email, password, phone = '', addresses = []}) {
    if (findUserByEmail(email)) {
      return { ok: false, message: 'Email already in use' };
    }
    // create new user with new id based on timestamp, role 'user'.
    const newUser = { id: `u${Date.now()}`, name, email, password, role:
'user', addresses, phone };
    setUsers((prev) => [newUser, ...prev]);
    setUser({ ...newUser, password: undefined });
    return { ok: true };
  }

  function login({ email, password }) {
    const u = findUserByEmail(email);
    if (!u) return { ok: false, message: 'User not found' };
    if (u.password !== password) return { ok: false, message: 'Invalid
password' };
    setUser({ ...u, password: undefined });
    return { ok: true };
  }

  function logout() {
    setUser(null);
  }

  function updateProfile(updates) {
    if (!user) return { ok: false, message: 'Not logged in' };
    setUsers((prev) => prev.map((u) => u.id === user.id ? { ...u,
...updates } : u));
    setUser((prev) => ({ ...prev, ...updates }));
    return { ok: true };
  }

  function addAddress(address) {
```

```
    if (!user) return { ok: false };
    const addr = { id: `a${Date.now()}`, ...address };
    setUsers((prev) => prev.map((u) => u.id === user.id ? { ...u,
addresses: [...(u.addresses || []), addr] } : u));
    setUser((prev) => ({ ...prev, addresses: [...(prev.addresses || []),
addr] }));
    return { ok: true };
  }

  function editAddress(id, updates) {
    if (!user) return { ok: false };
    setUsers((prev) => prev.map((u) => u.id === user.id ? { ...u,
addresses: (u.addresses || []).map((a) => a.id === id ? { ...a, ...updates
} : a) } : u));
    setUser((prev) => ({ ...prev, addresses: (prev.addresses ||
[]).map((a) => a.id === id ? { ...a, ...updates } : a) }));
    return { ok: true };
  }

  function removeAddress(id) {
    if (!user) return { ok: false };
    setUsers((prev) => prev.map((u) => u.id === user.id ? { ...u,
addresses: (u.addresses || []).filter((a) => a.id !== id) } : u));
    setUser((prev) => ({ ...prev, addresses: (prev.addresses ||
[]).filter((a) => a.id !== id) }));
    return { ok: true };
  }

  // Admin helpers
  function getAllUsers() {
    return users.map((u) => {
      const { password, ...rest } = u;
      return rest;
    });
  }

  function adminDeleteUser(id) {
    setUsers((prev) => prev.filter((u) => u.id !== id));
    // if current user deleted, logout
    if (user?.id === id) setUser(null);
  }

  const value = {
    user,
    users,
```

```
    signup,
    login,
    logout,
    updateProfile,
    addAddress,
    editAddress,
    removeAddress,
    getAllUsers,
    adminDeleteUser,
    isLoggedIn: !!user,
    isAdmin: user?.role === 'admin',
  };

  return <AuthContext.Provider
value={value}>{children}</AuthContext.Provider>;
}

export default AuthProvider;
```

```
//ProductContext.jsx
import React, { createContext, useEffect, useState } from 'react';
import sampleProducts from '../data/products';

export const ProductContext = createContext();

export function ProductProvider({ children }) {
  function dedupeById(arr) {
    const seen = new Map();
    for (const item of arr) {
      if (!seen.has(item.id)) seen.set(item.id, item);
    }
    return Array.from(seen.values());
  }

  const [products, setProducts] = useState(() => {
    try {
      const raw = localStorage.getItem('farm2door_products');
      let initial = raw ? JSON.parse(raw) : sampleProducts;

      // Merge with sampleProducts to ensure all products have correct
stock
      const sampleMap = new Map(sampleProducts.map((p) => [p.id, p]));
      initial = initial.map((p) => {
```

```
        const sample = sampleMap.get(p.id);
        // If product exists in sample, merge it (preserves cart changes
but resets stock if missing)
        if (sample) {
          return { ...p, stock: p.stock !== undefined ? p.stock :
sample.stock };
        }
        return p;
      });

      return dedupeById(initial);
    } catch {
      return dedupeById(sampleProducts);
    }
  });

  const [cart, setCart] = useState(() => {
    try {
      const raw = localStorage.getItem('farm2door_cart');
      return raw ? JSON.parse(raw) : [];
    } catch {
      return [];
    }
  });

  const [wishlist, setWishlist] = useState(() => {
    try {
      const raw = localStorage.getItem('farm2door_wishlist');
      return raw ? JSON.parse(raw) : [];
    } catch {
      return [];
    }
  });
// Orders placed by users
  const [orders, setOrders] = useState(() => {
    try {
      const raw = localStorage.getItem('farm2door_orders');
      return raw ? JSON.parse(raw) : [];
    } catch {
      return [];
    }
  });
// Search term for filtering products
  const [searchTerm, setSearchTerm] = useState('');
```

```
  // On mount, ensure any missing sample products are added (handles cases
  // where localStorage was previously trimmed). We merge only missing
items
  // by id so admin edits in localStorage are preserved.
  useEffect(() => {
    try {
      const currentIds = new Set(products.map((p) => p.id));
      const missing = sampleProducts.filter((p) => !currentIds.has(p.id));
      if (missing.length) {
        setProducts((prev) => {
          // Prepend missing sample products so they appear in the catalog
          // but keep existing (possibly admin-edited) products intact.
          const merged = [...missing, ...prev];
          // ensure uniqueness by id after merge
          return dedupeById(merged);
        });
      }
    } catch (err) {
      // silent
    }
    // run only once on mount
  }, []);
  useEffect(() => {
    try {
      const sampleMap = new Map(sampleProducts.map((p) => [p.id, p]));
      setProducts((prev) => {
        let changed = false;
        const next = prev.map((p) => {
          const sample = sampleMap.get(p.id);
          if (sample) {
            let updated = { ...p };
            if (p.image !== sample.image) {
              changed = true;
              updated.image = sample.image;
            }
            // Sync stock from sample products
            if (p.stock === undefined || p.stock < 0) {
              changed = true;
              updated.stock = sample.stock;
            }
            return updated;
          }
          return p;
        });
        return changed ? next : prev;
```

```
    });
  } catch (err) {
    // ignore
  }
  // run only once on mount
  // eslint-disable-next-line react-hooks/exhaustive-deps
}, []);

useEffect(() => {
  localStorage.setItem('farm2door_products', JSON.stringify(products));
}, [products]);

useEffect(() => {
  localStorage.setItem('farm2door_cart', JSON.stringify(cart));
}, [cart]);

useEffect(() => {
  localStorage.setItem('farm2door_wishlist', JSON.stringify(wishlist));
}, [wishlist]);

useEffect(() => {
  localStorage.setItem('farm2door_orders', JSON.stringify(orders));
}, [orders]);

function addToCart(product, qty = 1) {
  setCart((prev) => {
    // Get current stock from products state
    const currentProduct = products.find((p) => p.id === product.id);
    const availableStock = currentProduct?.stock || 0;

    // Prevent adding if no stock available
    if (availableStock <= 0) {
      return prev;
    }

    // Don't add more than available stock
    const qtyToAdd = Math.min(qty, availableStock);

    const found = prev.find((p) => p.id === product.id);
    if (found) {
      // Update quantities and reduce stock by the difference
      const newQty = found.qty + qtyToAdd;
      const qtyDifference = newQty - found.qty;

      // Reduce stock for the added quantity
```

```
      setProducts((prodsPrev) =>
        prodsPrev.map((p) =>
          p.id === product.id
            ? { ...p, stock: Math.max(0, p.stock - qtyDifference) }
            : p
        )
      );
      return prev.map((p) =>
        p.id === product.id ? { ...p, qty: newQty } : p
      );
    } else {
      // New item: reduce stock by qty
      setProducts((prodsPrev) =>
        prodsPrev.map((p) =>
          p.id === product.id
            ? { ...p, stock: Math.max(0, p.stock - qtyToAdd) }
            : p
        )
      );
      return [...prev, { ...product, qty: qtyToAdd }];
    }
  });
}

function updateCartQty(id, qty) {
  setCart((prev) => {
    const item = prev.find((p) => p.id === id);
    if (!item) return prev;

    const oldQty = item.qty;
    const newQty = Math.max(1, qty);
    const qtyDifference = newQty - oldQty;

    // Adjust stock based on quantity change
    setProducts((prodsPrev) =>
      prodsPrev.map((p) =>
        p.id === id
          ? { ...p, stock: Math.max(0, p.stock - qtyDifference) }
          : p
      )
    );

    return prev.map((p) =>
      p.id === id ? { ...p, qty: newQty } : p
    );
```

```
    });
  }

  function removeFromCart(id) {
    setCart((prev) => {
      const item = prev.find((p) => p.id === id);
      if (item) {
        // Restore stock when item is removed
        setProducts((prodsPrev) =>
          prodsPrev.map((p) =>
            p.id === id
              ? { ...p, stock: p.stock + item.qty }
              : p
          )
        );
      }
      return prev.filter((p) => p.id !== id);
    });
  }

  function addToWishlist(product) {
    setWishlist((prev) => {
      if (prev.find((p) => p.id === product.id)) return prev;
      return [...prev, product];
    });
  }

  function removeFromWishlist(id) {
    setWishlist((prev) => prev.filter((p) => p.id !== id));
  }

  function adminAddProduct(newProduct) {
    setProducts((prev) => [{ ...newProduct, id: `p${Date.now()}` },
...prev]);
  }

  function adminEditProduct(updated) {
    setProducts((prev) => prev.map((p) => p.id === updated.id ? { ...p,
...updated } : p));
  }

  function adminDeleteProduct(id) {
    setProducts((prev) => prev.filter((p) => p.id !== id));
    // also remove from cart/wishlist if present
    setCart((prev) => prev.filter((c) => c.id !== id));
```

```
    setWishlist((prev) => prev.filter((w) => w.id !== id));
  }

  function resetCatalog() {
    // Replace products with the original sampleProducts.
    // This is intended for demo/admin to restore defaults.
    setProducts(sampleProducts);
  }

  function createOrder({ items, total, user }) {
    const order = {
      id: `o${Date.now()}`,
      items,
      total,
      user: user || null,
      date: new Date().toISOString(),
    };
    setOrders((prev) => [order, ...prev]);
    // clear cart after order
    setCart([]);
    return order;
  }

  const filtered = products.filter((p) =>
p.title.toLowerCase().includes(searchTerm.toLowerCase()) ||
p.category.toLowerCase().includes(searchTerm.toLowerCase()));

  const value = {
    products,
    filtered,
    cart,
    wishlist,
    orders,
    searchTerm,
    setSearchTerm,
    addToCart,
    updateCartQty,
    removeFromCart,
    addToWishlist,
    removeFromWishlist,
    adminAddProduct,
    adminEditProduct,
    adminDeleteProduct,
    resetCatalog,
    createOrder,
```

```
  };

  return <ProductContext.Provider
value={value}>{children}</ProductContext.Provider>;
}

export default ProductProvider;
```

8. Installation & Setup

Prerequisites: Node.js (LTS), npm or yarn.

Install: run "npm install" to install dependencies.

Run locally: "npm run dev" or "npm start"; open http://localhost:5173 or your configured port.

Build/Deploy: "npm run build"; serve the build output via a static host or integrate with your backend.

9. Testing

Test Coverage: target high coverage for contexts and critical UI paths; include negative cases (invalid login, empty cart, Required Validations,Checkout).

10. Security Considerations

Authentication/Authorization: protect routes with guards (requireAuth, requireAdmin); enforce role checks in UI and API.

Password Hashing: store hashed passwords (bcrypt/Argon2) when backend is enabled; never store plain text.

Data Storage Practices: validate inputs, avoid sensitive data in localStorage, use HTTPS in production, and implement rate limiting on auth endpoints.

11. Future Improvements

Planned Features: payment gateway integration, order tracking, reviews/ratings, email notifications.

Scalability: move persistence to backend/cloud (e.g., REST API, managed database); introduce caching and pagination for large catalogs.

Migration: deploy on cloud services (Azure/AWS/GCP), add CI/CD and monitoring.