MIT Art, Design and Technology University
MIT School of Computing, Pune

Department of Information Technology

SUBJECT – FULL STACK FRONT END DEVELOPMENT

COURSE CODE – 23IT3001

Class – T.Y. SMAD

Name of Student

Ibrahim Poonawala - ADT23SOCB1523

Under the Guidance of

Prof. Rohini Bhosale

A.Y. 2025-2026 (SEM – I)

Assignment 1 : React Project Setup and Navigation

Aim :

To create a React project using create-react-app, set up routing using react-router-dom, and build a top navigation bar with links to different pages such as Home, Products, Cart, Checkout, and Profile.

Theory :

- **Introduction to React**

  React is an open-source JavaScript library developed by Facebook for building user interfaces, especially for single-page applications. It allows developers to create reusable UI components, manage data efficiently, and render the user interface dynamically without reloading the entire page. It follows a component-based architecture, meaning that every part of the UI is broken down into small, independent, and reusable components.
  React uses a virtual DOM (Document Object Model) which enhances performance by updating only the parts of the web page that have changed, rather than re-rendering the whole page.

- **React Project Setup using create-react-app**

  To begin working with React, developers use the create-react-app tool. It is an officially supported way to set up a new React project quickly without configuring build tools like Webpack or Babel manually.

  Key Features of create-react-app:

  1. Automatically configures the project structure.
  2. Includes a development server with live reloading.
  3. Simplifies dependency management.
  4. Provides built-in support for JSX and ES6 syntax.

  After running the setup command, the tool generates all the necessary files and dependencies required for a React environment.

- **React Router and Single Page Applications (SPA)**

  In traditional websites, each link click results in a new page request to the server. React, however, uses Single Page Application (SPA) architecture, where only one HTML page is loaded, and the content changes dynamically using JavaScript.
  To handle multiple views or pages within an SPA, React developers use a library called react-router-dom.
  React Router enables navigation between different components in the application without refreshing the page. It provides components like:

---

1. BrowserRouter: Wraps the whole application and keeps the UI in sync with the URL.
2. Routes and Route: Define which component should render for a specific path.
3. Link and NavLink: Used to navigate between pages without reloading.

- **Navigation Bar (Navbar)**

  A Navigation Bar is an essential user interface element that allows users to move between different sections of the application easily.

  In a React application, the navigation bar is typically created as a separate reusable component. It includes several navigation links (using the Link component from React Router) that point to different routes defined in the application.

  Features of a Good Navigation Bar:

  1. It is consistent across all pages.

  2. It provides clear and visible links to major sections.

  3. It is responsive, meaning it adjusts to different screen sizes.

  4. It often includes a brand logo or title on the left and menu options on the right.

- **Components and Routing Structure**

  In a React project, each webpage or screen is generally represented as a component. Examples of such components include Home, Products, Cart, Checkout, and Profile. Each of these components is connected using React Router routes.

  When the user clicks a link in the navigation bar, the URL changes and the corresponding component is rendered — all without reloading the page.

  This creates a smooth and seamless navigation experience for the user.

- **Working of Navigation in React**

  1. The application is wrapped inside a **Router** (BrowserRouter).
  2. Each page or screen is defined as a **Route**.
  3. The **Navbar component** contains clickable links to these routes.
  4. When a link is clicked, React Router intercepts the navigation event and updates the view accordingly.
  5. The **current component** associated with the selected route is displayed dynamically.

  This approach provides faster navigation, better user experience, and efficient content loading.

- **Advantages of Using React Router and Navigation**

  1. Improved User Experience: Page transitions are instant without full reloads.

2. SEO Friendly: Routes can be defined clearly with specific URLs.
3. Component-Based Organization: Each route corresponds to a specific component, making maintenance easier.
4. Scalability: New pages and routes can be added easily as the application grows.
5. Performance: Navigation happens on the client-side, reducing server load.

Code :

npm create vite@latest create-app npm

install

cd create-app

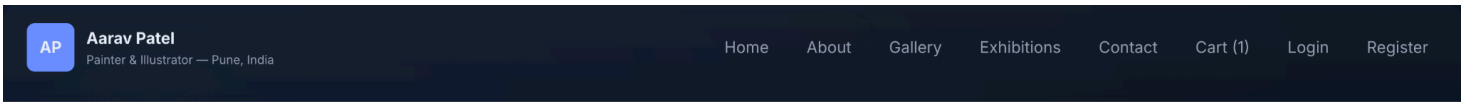
npm run dev

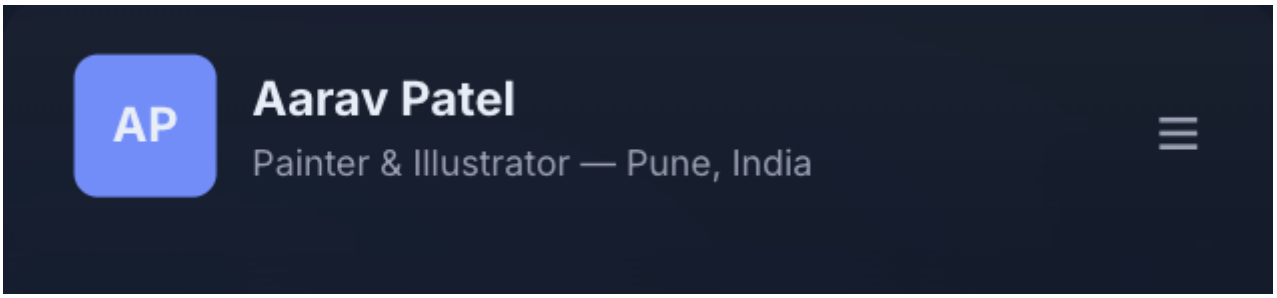
```
// src/components/Header.jsx
import React, {useEffect, useState} from
"react";
import { NavLink, useNavigate } from
"react-router-dom";
import { getCurrentUser, logout } from
"../utils/auth";
import { cartCount } from "../utils/cart";

export default function Header(){
 const nav = useNavigate();
 const [user, setUser] =
useState(getCurrentUser());
 const [count, setCount] =
useState(cartCount());

 useEffect(()=> {
   // simple polling of localStorage
changes (demo). If you implement events,
use that.
   const id = setInterval(()=> {
setUser(getCurrentUser());
setCount(cartCount()); }, 500);
   return ()=> clearInterval(id);
 }, []);

 function handleLogout(){
   logout();
   setUser(null);
   nav("/");
 }

 return (
   <header className="header">
     <div className="logo">
       <div className="mark">AP</div>
       <div>
         <div
style={{fontWeight:700}}>Aarav Patel</div>
         <div
style={{fontSize:12,color:'var(--muted)'}}
>Painter & Illustrator — Pune, India</div>
       </div>
     </div>

     <nav className="nav">
       <NavLink to="/" end>Home</NavLink>
       <NavLink
to="/about">About</NavLink>
       <NavLink
to="/gallery">Gallery</NavLink>
       <NavLink
to="/exhibitions">Exhibitions</NavLink>
       <NavLink
to="/contact">Contact</NavLink>
       <NavLink to="/cart">Cart
```

```
({count})</NavLink>
        {user ? (
          <>
            <span
style={{color:'var(--muted)',paddingLeft:8
}}>Hi, {user.name}</span>
            <button className="btn
btn-outline"
onClick={handleLogout}>Logout</button>
          </>
        ) : (
          <>
```

```
            <NavLink
to="/login">Login</NavLink>
            <NavLink
to="/register">Register</NavLink>
          </>
        )}
      </nav>

      <div className="hamburger">☰</div>
    </header>
  );
}
```

Output :





Conclusion :

We successfully implement Navbar using jsx.

Assignment 2 : Home Page Development

Aim :

To design and develop the Home Page layout of a React application that includes featured products or categories and a functional search bar that filters products by name or category.

Theory :

- **Introduction**

  The Home Page is the first and most important part of any website or web application. It provides an overview of the application's purpose, displays key features or products, and helps users easily navigate to other sections.

  In this assignment, the Home Page is developed using React components, Bootstrap for styling, and includes dynamic rendering of featured products.

- **Concept of Components in React**

  React follows a component-based architecture, which means the UI is built using independent, reusable pieces of code called components.

  In this assignment:

  The Home.js file acts as a page-level component.

  ProductCard.js acts as a reusable subcomponent for displaying product information such as name, category, and price.

  Each component has its own logic and UI, allowing separation of concerns and easier maintenance.

- **Designing the Home Page Layout**

  A Home Page layout typically includes:
  1. A banner or header section to welcome the user.
  2. A featured products section showing important or popular items.
  3. A search bar to help users find products quickly.
  4. Proper alignment and spacing using Bootstrap classes such as container, row, and col.

  This helps in creating a responsive and clean layout suitable for all devices.

- **Featured Products or Categories**

The featured products section highlights key products or categories to attract user attention.

In React, this is implemented by maintaining a list (array) of products inside the component's state. Each product has attributes such as:

- id

- name

- category

- price

- image

These products are then dynamically displayed using the map() function.

This ensures scalability — new products can be added easily without modifying the UI code.

- **Styling and Responsiveness**

Bootstrap and CSS are used to style the page:

- Containers and grid layouts organize products in rows and columns.

- The navigation bar remains fixed.

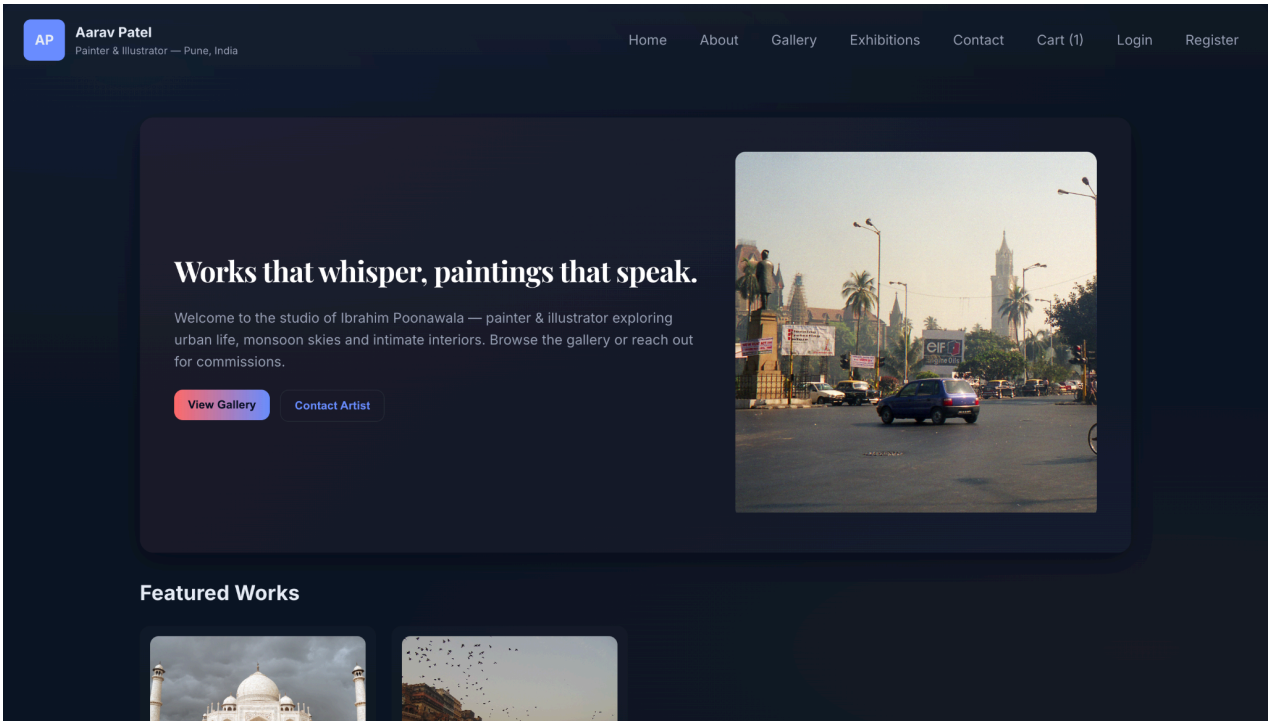
- Margins and padding ensure spacing between elements.

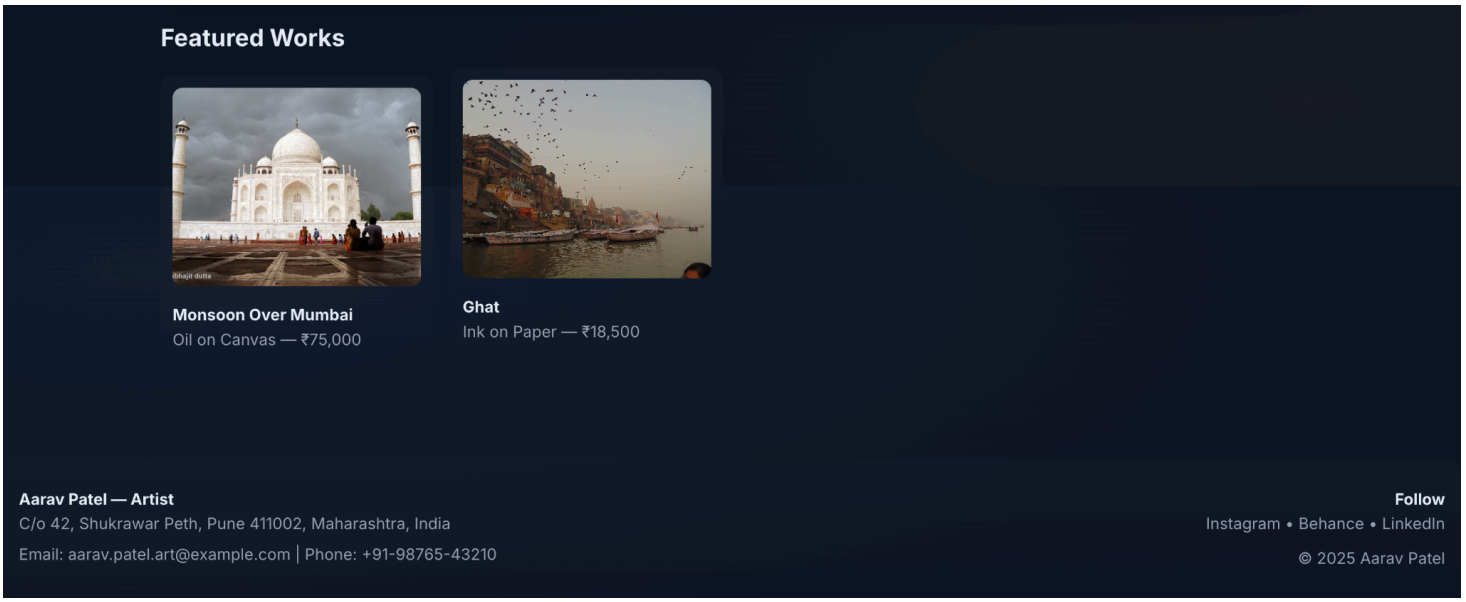
- Responsive design ensures the layout adapts to different screen sizes

Code :

```
import React from 'react'
import { Link } from 'react-router-dom'

export default function Home(){
 return (
   <section>
     <div className="hero">
       <div>
         <h1>Works that whisper, paintings that speak.</h1>
         <p>Welcome to the studio of Ibrahim Poonawala — painter & illustrator exploring urban life, monsoon skies and intimate interiors. Browse the gallery or reach out for commissions.</p>
         <div className="cta">
           <Link to="/gallery"><button className="btn btn-primary">View Gallery</button></Link>
           <Link to="/contact"><button className="btn btn-outline">Contact Artist</button></Link>
         </div>
       </div>
       <div>
         <div style={{borderRadius:12,overflow:'hidden'}}>
           <img src="public/assets/artworks/mumbai-street.jpg" alt="Featured artwork" style={{width:'100%',height:420,objectFit:'cover'}} />
         </div>
       </div>
     </div>

     <h2 style={{marginTop:28}}>Featured Works</h2>
     <div style={{marginTop:16}} className="gallery-grid">
       {/* We'll show a few manual featured cards for the Home page — in production map from data */}
       <article className="card">
         <img src="public/assets/artworks/monsoon-india.jpg" alt="Monsoon" />
         <div className="meta"><strong>Monsoon Over Mumbai</strong><div style={{color:'var(--muted)'}}>Oil on Canvas — ₹75,000</div></div>
       </article>
       <article className="card">
         <img src="public/assets/artworks/varanasi-ghat.jpg" alt="Ghat" />
         <div className="meta"><strong>Ghat</strong><div style={{color:'var(--muted)'}}>Ink on Paper — ₹18,500</div></div>
       </article>
     </div>
   </section>
 )
}
```

Output :





Conclusion :

We successfully implemented Home page , Footer Functionality.

Assignment 3 : Fetch product data from array or static JSON file

---

Aim :

To understand and implement Routing in React using the react-router-dom library, and to design a Products page that displays a list of 12 products with their details and an option to add them to the cart.

Theory :

- **Introduction to Routing**

  Routing in React refers to the process of navigating between different views or pages in a single-page application (SPA).

  In traditional websites, navigation between pages requires full-page reloads, but React applications handle navigation on the client-side using JavaScript — resulting in a faster, smoother user experience.

  Routing in React is managed through the React Router library (react-router-dom).

- **What is React Router ?**

  React Router is a powerful and widely used library for managing routes in React applications. It allows you to define different paths (URLs) and map them to specific components that should render when those paths are accessed.

- **Components of React Router**

Component	Description
BrowserRouter	The main container that keeps the UI in sync with the URL. It wraps the entire app.
Routes	Acts as a container for all route definitions.
Route	Defines a specific path and the component that should render when the path is matched.
Link	Replaces <a> tags for navigation. Prevents page reload and provides smooth transitions.
NavLink	Similar to Link, but adds active styling to indicate the current route.

useNavigate	A hook used to programmatically navigate between routes.

- **How Routing Works ?**

1. The entire React application is wrapped inside <BrowserRouter>.
2. The <Routes> component defines multiple <Route> components.
3. Each <Route> specifies a URL path and the corresponding component to render.
4. When the user clicks a Link (or navigates to a URL), React Router dynamically renders the corresponding component without reloading the page.

This approach gives the illusion of multiple pages while actually rendering everything within one HTML document — hence the term Single Page Application.

- **Advantages of Routing**

- Fast Navigation: No page reloads; routing handled client-side.
- Better User Experience: Seamless transitions between pages.
- Maintainability: Each route is mapped to a separate, modular component.
- Dynamic Routes: Pages can be rendered based on dynamic data (e.g., /products/:id).
- Reusable Components: Navigation structure remains the same while content changes dynamically.

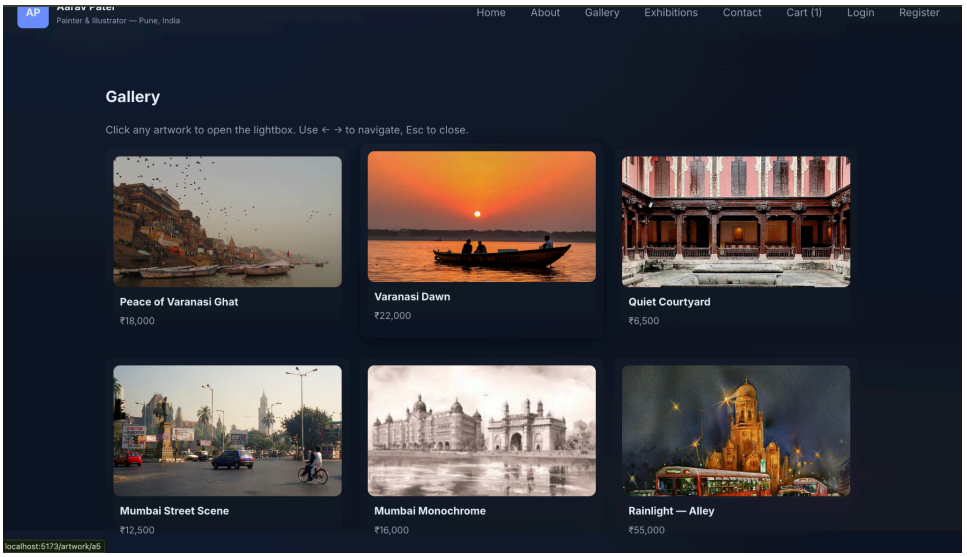
Code :

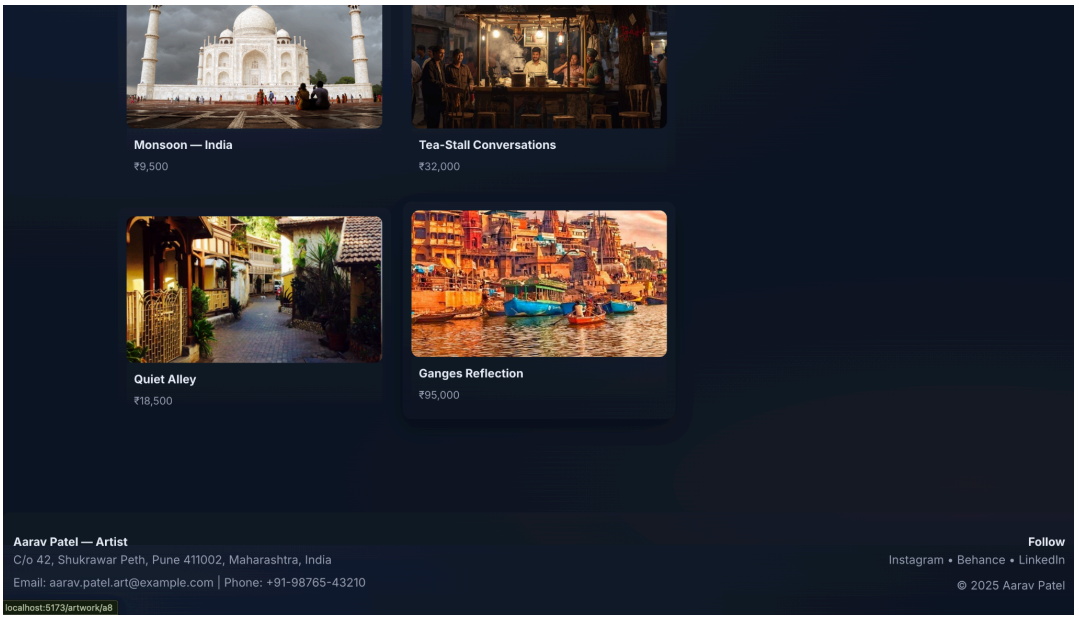
```
[
 {
   "id": "a1",
   "title": "Peace of Varanasi Ghat",
   "year": "2016",
   "medium": "Photography (Commons)",
   "dimensions": "various",
   "description": "A calm morning at the ghats of Varanasi — sourced from Wikimedia Commons (see attribution on file page).",
   "images": ["/assets/artworks/varanasi-ghat.jpg"],
   "tags": ["Photography", "Spiritual", "India"],
   "price": "₹18,000",
   "availability": "Available",
   "location": "Varanasi, Uttar Pradesh, India",
   "source": "Wikimedia Commons — VARANASI_GHAT_,_VARANASI.jpg"
 },
 {
   "id": "a2",
   "title": "Mumbai Street Scene",
   "year": "2011",
   "medium": "Photography (Commons, CC BY 2.0)",
   "dimensions": "various",
   "description": "Street life in Mumbai captured in a candid frame — Creative Commons image (attribution required: see file page).",
   "images": ["/assets/artworks/mumbai-street.jpg"],
   "tags": ["Photography", "Urban", "India"],
   "price": "₹12,500",
   "availability": "Available",
```

```
    "location": "Mumbai, Maharashtra, India",
    "source": "Wikimedia Commons — Mumbai_street_scene_(6409592653).jpg"
  },
  {
    "id": "a3",
    "title": "Monsoon — India",
    "year": "2009",
    "medium": "Photography (Commons)",
    "dimensions": "various",
    "description": "Monsoon mood captured — sourced from Wikimedia Commons.",
    "images": ["/assets/artworks/monsoon-india.jpg"],
    "tags": ["Photography", "Monsoon", "Nature"],
    "price": "₹9,500",
    "availability": "Available",
    "location": "India",
    "source": "Wikimedia Commons — Monsoon_in_india.jpg"
  } ,
{
  "id": "a4",
    "title": "Quiet Alley",
    "year": "2023",
    "medium": "Ink on Paper",
    "dimensions": "18 x 24 in",
    "description": "A study in light and shadow from the old quarters.",
    "images": ["/assets/artworks/quiet-alley-1.jpg"],
    "tags": ["Drawing", "Ink", "Urban"],
    "price": "₹18,500",
    "availability": "Sold",
    "location": "Mumbai, Maharashtra, India"
} ,
{
  "id": "a5",
    "title": "Varanasi Dawn",
    "year": "2022",
    "medium": "Watercolor on Paper",
    "dimensions": "20 x 26 in",
    "description": "Soft morning light over the ghats captured in careful washes of color.",
    "images": ["/assets/artworks/varanasi-dawn.jpg"],
    "tags": ["Watercolor", "Spiritual", "Landscape"],
    "price": "₹22,000",
    "availability": "Available",
    "location": "Varanasi, Uttar Pradesh, India"
},
{
  "id": "a6",
    "title": "Mumbai Monochrome",
    "year": "2021",
    "medium": "Charcoal on Paper",
    "dimensions": "24 x 30 in",
    "description": "Monochrome study of a busy street corner — emphasis on line and contrast.",
    "images": ["/assets/artworks/mumbai-monochrome.jpg"],
    "tags": ["Charcoal", "Urban"],
    "price": "₹16,000",
    "availability": "Available",
    "location": "Mumbai, Maharashtra, India"
},
{
  "id": "a7",
    "title": "Tea-Stall Conversations",
    "year": "2024",
    "medium": "Acrylic on Board",
    "dimensions": "30 x 22 in",
    "description": "Everyday interactions in the neighbourhood translated into warm strokes.",
    "images": ["/assets/artworks/tea-stall.jpg"],
    "tags": ["Acrylic", "Figurative", "India"],
```

```
    "price": "₹32,000",
    "availability": "Available",
    "location": "Pune, Maharashtra, India"
},
{
  "id": "a8",
    "title": "Ganges Reflection",
    "year": "2020",
    "medium": "Oil on Canvas",
    "dimensions": "40 x 30 in",
    "description": "Large canvas study of
reflections and ritual movement on the
Ganges.",
    "images":
["/assets/artworks/ganges-reflection.jpg"],
    "tags": ["Oil", "Spiritual", "Large"],
    "price": "₹95,000",
    "availability": "Available",
    "location": "Varanasi, Uttar Pradesh, India"
  },
{
"id": "a9",
    "title": "Quiet Courtyard",
    "year": "2023",
    "medium": "Digital Print (Limited Edition)",
    "dimensions": "12 x 16 in",
    "description": "Limited edition archival
print of a small courtyard study (signed).",
    "images":
["/assets/artworks/quiet-courtyard.jpg"],
    "tags": ["Print", "Limited", "Urban"],
    "price": "₹6,500",
    "availability": "Available",
    "location": "Pune, Maharashtra, India"
},
{
  "id": "a10",
    "title": "Rainlight — Alley",
    "year": "2024",
    "medium": "Oil on Canvas",
    "dimensions": "30 x 40 in",
    "description": "Dramatic contrasts and wet
pavement reflections during monsoon evenings.",
    "images":
["/assets/artworks/rainlight-alley.jpg"],
    "tags": ["Oil", "Monsoon"],
    "price": "₹55,000",
    "availability": "Available",
    "location": "Mumbai, Maharashtra, India"
}
]
```

Output :





Conclusion :

We successfully implemented JSON handling and routing.

Assignment 4 : Products Page

Aim :

To design and develop a Products Page in React that dynamically displays a list of products with complete information such as image, name, description, price, and quantity, along with an “Add to Cart” button. The page also includes a functional search bar to filter products based on name or description.

Theory :

- **Introduction**

  The Products Page serves as the core section of any e-commerce or healthcare-related application.

  It lists all available items in an organized layout so users can browse, view details, and add products to their cart.

  In a React application, this page is implemented as a functional component that manages data using state and props, and utilizes the component-based architecture for code reusability.

- **Component-Based Architecture**

  React follows a modular design approach where the UI is divided into independent, reusable components.
  For the Products Page:

  - Products.js acts as the *page-level* component responsible for logic and filtering.

  - ProductCard.js acts as a *presentational subcomponent* displaying a single product’s details.

  This structure makes the app scalable and easy to maintain — any future change in product presentation requires editing only one file.

- **Featured Products or Categories**

  The featured products section highlights key products or categories to attract user attention.

  In React, this is implemented by maintaining a list (array) of products inside the component’s state.
  Each product has attributes such as:

  - id

  - name

  - category

- price

- image

These products are then dynamically displayed using the map() function.

This ensures scalability — new products can be added easily without modifying the UI code.

- **Routing and Navigation**

Navigation to the Products Page happens through the Navbar using the React Router <Link> component.

When a user clicks “Products” in the navigation bar or on a product in the Home Page, they are routed to /products without reloading the entire page.

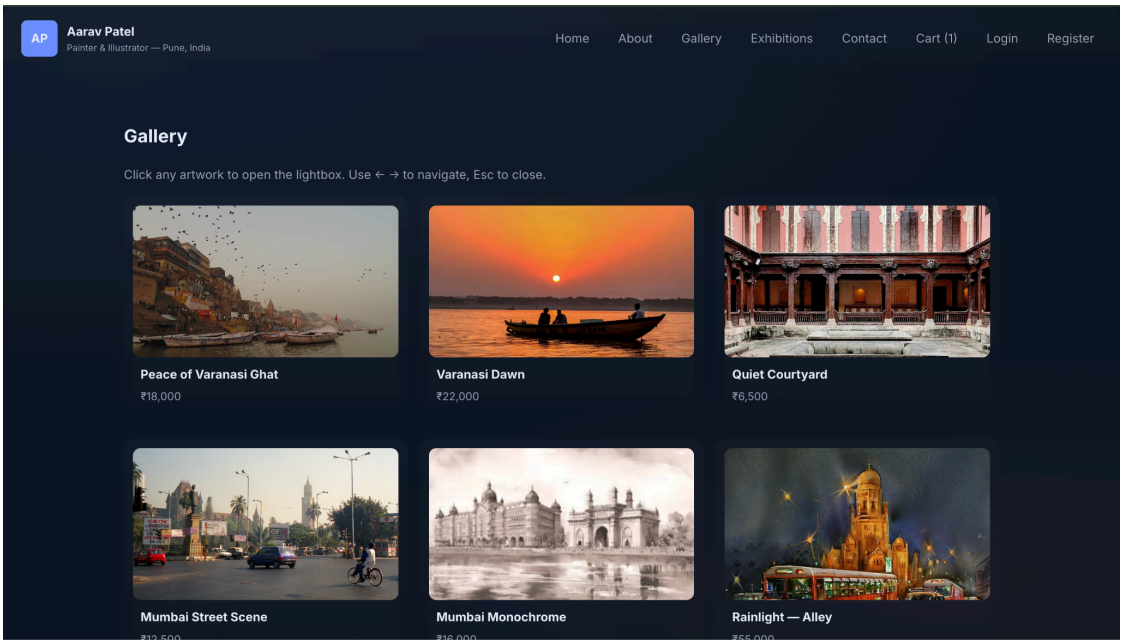
This client-side routing creates a seamless experience typical of Single Page Applications (SPA).

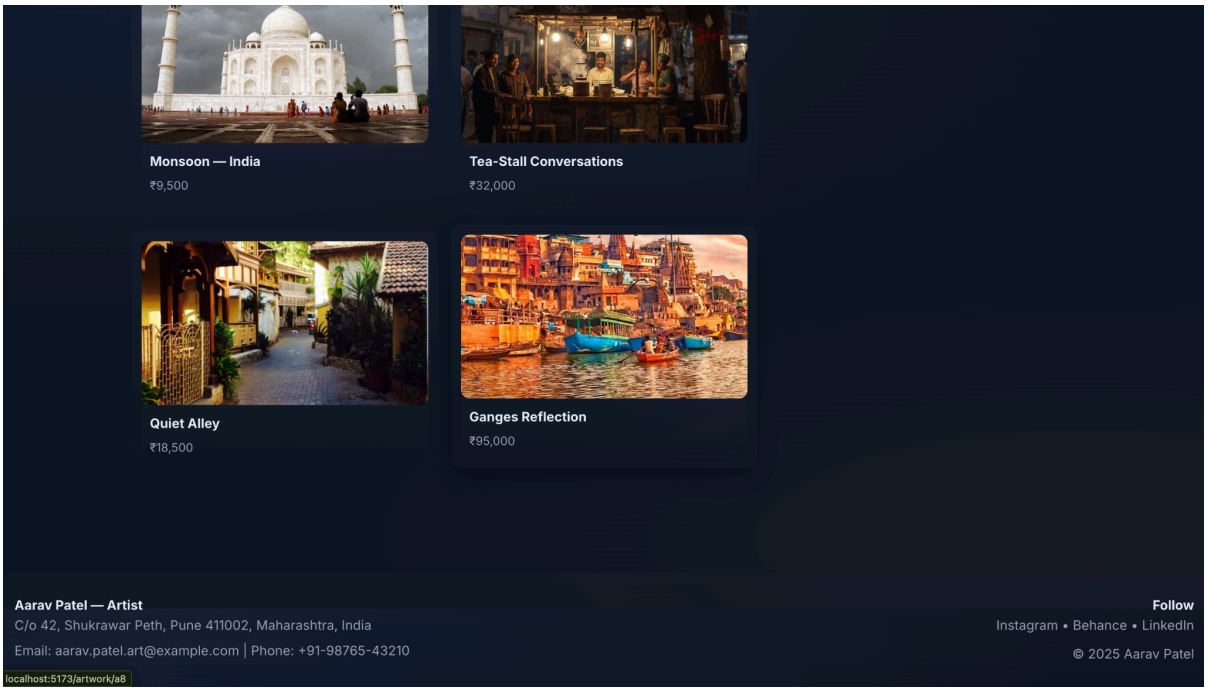
Code :

```
// src/pages/ArtworkDetails.jsx
import React, { useEffect, useState }
from "react";
import { useParams, useNavigate } from
"react-router-dom";
import { addToCart } from
"../utils/cart";

export default function ArtworkDetails()
{
 const { id } = useParams();
 const navigate = useNavigate();
 const [artwork, setArtwork] =
useState(null);
 const [loading, setLoading] =
useState(true);
 const [error, setError] = useState("");

 useEffect(() => {
   async function fetchArtwork() {
     try {
       const res = await
fetch("/artworks.json"); // must be in
/public/
       if (!res.ok) throw new
Error(`HTTP ${res.status}`);
       const list = await res.json();
       const found = list.find((a) =>
a.id === id);
       if (!found) throw new
Error("Artwork not found");
       setArtwork(found);
     } catch (err) {
       console.error("Error loading
artwork:", err);
       setError("Could not load artwork
details.");
     } finally {
       setLoading(false);
     }
   }
   fetchArtwork();
 }, [id]);

 const handleAddToCart = () => {
   addToCart({
     id: artwork.id,
     title: artwork.title,
     price: artwork.price,
     image: artwork.images?.[0],
   });
   alert(`✅ "${artwork.title}" added to
cart!`);
 };
```

```
 const handleBuyNow = () => {
   addToCart({
     id: artwork.id,
     title: artwork.title,
     price: artwork.price,
     image: artwork.images?.[0],
   });
   navigate("/cart");
 };

 if (loading) return <p style={{
textAlign: "center" }}>Loading...</p>;
 if (error) return <p style={{ color:
"red", textAlign: "center"
}}>{error}</p>;

 return (
   <section style={{ padding: "2rem",
textAlign: "center" }}>
     <h2>{artwork.title}</h2>
     <img
       src={artwork.images?.[0]}
       alt={artwork.title}
       style={{ maxWidth: "600px",
borderRadius: "10px", marginTop: "1rem"
}}
     />
     <p style={{ marginTop: "1rem",
fontSize: "1.2rem"
}}>{artwork.description}</p>
     <p style={{ fontSize: "1.4rem",
color: "var(--accent)", marginTop:
"1rem" }}>
       Price: {artwork.price}
     </p>

     <div style={{ marginTop: "2rem" }}>
       <button
         onClick={handleAddToCart}
         style={{
           backgroundColor: "#22c55e",
           color: "#fff",
           border: "none",
           borderRadius: "6px",
           padding: "10px 20px",
           marginRight: "10px",
           cursor: "pointer",
         }}
       >
         Add to Cart
       </button>
       <button
         onClick={handleBuyNow}
         style={{
           backgroundColor: "#2563eb",
           color: "#fff",
           border: "none",
           borderRadius: "6px",
           padding: "10px 20px",
           cursor: "pointer",
         }}
       >
         Buy Now
       </button>
     </div>
   </section>
 );
}
```

Output :


AP
Aarav Patel
Painter & Illustrator — Pune, India
Home
About
Gallery
Exhibitions
Contact
Cart (1)
Login
Register
Gallery
Click any artwork to open the lightbox. Use ← → to navigate, Esc to close.
Peace of Varanasi Ghat
₹18,000
Varanasi Dawn
₹22,000
Quiet Courtyard
₹6,500
Mumbai Street Scene
Mumbai Monochrome
Rainlight — Alley


Monsoon — India
₹9,500
Tea-Stall Conversations
₹32,000
Quiet Alley
₹18,500
Ganges Reflection
₹95,000
Aarav Patel — Artist
C/o 42, Shukrawar Peth, Pune 411002, Maharashtra, India
Email: aarav.patel.art@example.com | Phone: +91-98765-43210
Follow
Instagram • Behance • LinkedIn
© 2025 Aarav Patel
localhost:5173/artwork/a8

Conclusion :

We successfully implemented Products Page.

Assignment 5 : Cart Page

Aim :

To design and develop a Cart Page in React that allows users to view, manage, and modify selected products added from the Products page.

Theory :

- **Introduction**

  The Cart Page is a crucial component of any e-commerce or online pharmacy application.

  It acts as a temporary storage area where selected products are displayed before checkout.

  This page provides functionalities such as updating quantity, deleting items, and viewing the total price dynamically — all managed through React state.

- **Data Flow in React Cart Page**

  The ProductCard component passes product details to the Cart Page via React Router's navigate() function and useLocation() hook.

  When a user clicks "Add to Cart", the selected product data is sent as a state object to /cart.

- **State Management**

  React's useState() hook is used to store and manage all cart items.

- **Component Structure**

  1. ProductCard.js: Contains "Add to Cart" button.
  2. Cart.js: Displays all selected products with options to update or remove.
  3. App.js: Handles routing between /products and /cart.

  This modular design ensures code reusability, scalability, and separation of concerns.

- **Quantity Handling**

  Each product's quantity is controlled by an input field.

- **Deletion of Products**

  The delete functionality uses the filter() method.\

Code :

```
// src/pages/Cart.jsx

import React, { useEffect, useState }
from "react";

import { useNavigate } from
"react-router-dom";

import { getCurrentUser } from
"../utils/auth";

import { getCart, removeFromCart,
updateQty, clearCart, parseINR } from
"../utils/cart";



export default function Cart() {

 const nav = useNavigate();

 const [user, setUser] =
useState(getCurrentUser());

 const [cart, setCart] =
useState(getCart());

 const [message, setMessage] =
useState("");



 useEffect(() => {

   // update state from localStorage
on mount

   setUser(getCurrentUser());

   setCart(getCart());

   // simple interval to reflect
external changes (e.g., addToCart
from other page)

   const id = setInterval(() =>
setCart(getCart()), 500);

   return () => clearInterval(id);

 }, []);



 function handleRemove(id) {

   const updated =
removeFromCart(id);

   setCart(updated);

 }



 function handleQtyChange(id, qty) {

   if (qty < 1) qty = 1;

   updateQty(id, qty);

   setCart(getCart());

 }



 function handleClear() {

   clearCart();

   setCart([]);

 }



 function handleCheckout() {

   const currentUser =
getCurrentUser();

   if (!currentUser) {

     setMessage("Please login or
```

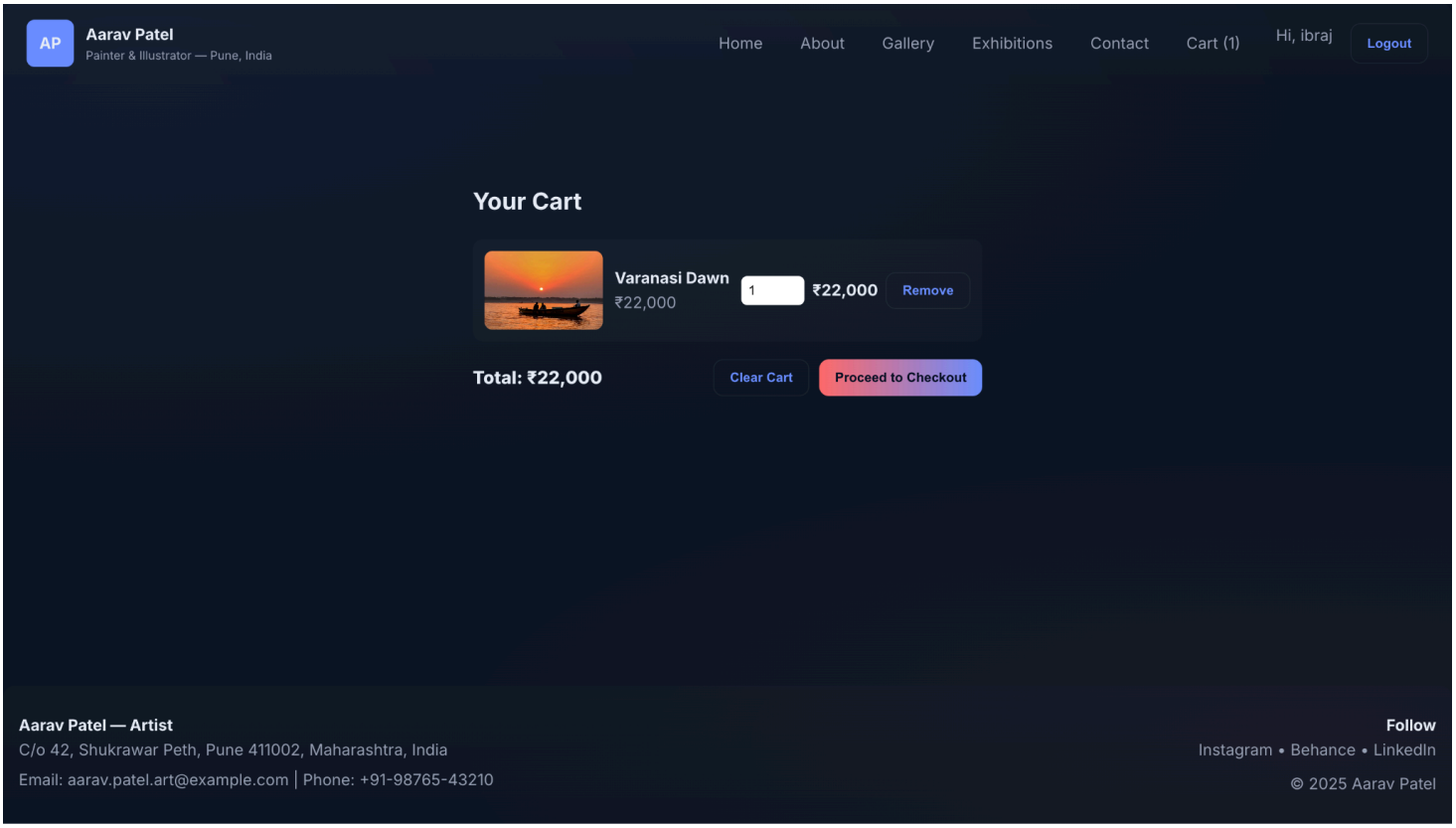
```
register before checking out.");
      return;
    }
    if (cart.length === 0) {
      setMessage("Your cart is
empty.");
      return;
    }

    // compute numeric total (parseINR
turns "₹12,000" -> 12000)
    const totalNumber =
cart.reduce((s, i) => s +
parseINR(i.price) * (i.qty || 1), 0);

    // Simulate a payment flow (demo)
    // For demo: show a fake UPI/QR
modal or simple confirmation
    const successMsg = `Payment
simulated:
₹${totalNumber.toLocaleString("en-IN"
)} charged to ${currentUser.email}.
Thank you for your purchase!`;
    setMessage(successMsg);

    // clear cart after checkout
    clearCart();
    setCart([]);
  }

  // If not logged in, show a polite
login prompt (but still allow view if
you prefer)
  if (!user) {
    return (
      <div style={{ padding: 40,
textAlign: "center" }}>
        <h2>Please login to view your
cart.</h2>
        <button
          onClick={() =>
nav("/login")}
          style={{ padding: "10px
18px", borderRadius: 8, background:
"linear-gradient(90deg,#ff7b7b,#7b9ef
f)", color: "#051023", border:
"none", cursor: "pointer" }}
        >
          Go to Login
        </button>
      </div>
    );
  }

  if (cart.length === 0) {
    return (
      <div style={{ padding: 40 }}>
        <h2>Your cart is empty</h2>
        <p style={{ color:
"var(--muted)" }}>Browse the gallery
and add artworks to buy.</p>
      </div>
    );
  }

  const total = cart.reduce((s, i) =>
s + parseINR(i.price) * (i.qty || 1),
0);
```

```
 return (
   <section style={{ padding: 32,
maxWidth: 1000, margin: "0 auto" }}>
     <h2>Your Cart</h2>
     <div style={{ display: "grid",
gap: 12 }}>
       {cart.map((item) => (
         <div key={item.id} style={{
display: "flex", gap: 12, alignItems:
"center", padding: 12, background:
"rgba(255,255,255,0.02)",
borderRadius: 10 }}>
           <img src={item.image}
alt={item.title} style={{ width: 120,
height: 80, objectFit: "cover",
borderRadius: 8 }} />
           <div style={{ flex: 1 }}>
             <div style={{
fontWeight: 700 }}>{item.title}</div>
             <div style={{ color:
"var(--muted)" }}>{item.price}</div>
           </div>
           <div style={{ display:
"flex", gap: 8, alignItems: "center"
}}>
             <input
               type="number"
               min={1}
               max={10}
               value={item.qty || 1}
               onChange={(e) =>
handleQtyChange(item.id,
Number(e.target.value))}
               style={{ width: 64,
padding: 6, borderRadius: 6, border:
"1px solid rgba(255,255,255,0.06)" }}
             />
             <div style={{
fontWeight: 700
}}>₹{(parseINR(item.price) *
(item.qty ||
1)).toLocaleString("en-IN")}</div>
             <button className="btn
btn-outline" onClick={() =>
handleRemove(item.id)}>Remove</button
>
           </div>
         </div>
       ))}
     </div>
     <div style={{ marginTop: 18,
display: "flex", justifyContent:
"space-between", alignItems: "center"
}}>
       <div style={{ fontWeight: 800,
fontSize: 18 }}>Total:
₹{total.toLocaleString("en-IN")}</div
>
       <div style={{ display: "flex",
gap: 10 }}>
         <button className="btn
btn-outline"
onClick={handleClear}>Clear
Cart</button>
         <button className="btn
btn-primary"
onClick={handleCheckout}>Proceed to
Checkout</button>
```

```
      </div>

    </div>


    {message && <div style={{
marginTop: 12, color: "lightgreen"
```

```
}}>{message}</div>}

  </section>

 );

}
```

Output:



Conclusion:

We successfully implemented Cart Page.

Assignment 6 : Checkout Page

Aim :

To create a Checkout Page that collects user shipping information and displays an Order Summary, allowing users to confirm their purchase through a Place Order button.

Theory :

- **Introduction**

  The Checkout Page is the final step of the online purchase process.
  It allows users to provide necessary shipping details, review their order summary, and confirm the order.
  This page integrates form handling, data validation, and display of cart data using React state and React Router.

- **Role of Checkout Page**

  The Checkout Page serves three primary purposes:

  1. Collecting User Details – It captures customer information such as name, address, city, postal code, and contact number.
  2. Reviewing Order Summary – It provides a detailed summary of all products added to the cart, including their quantities and total price.
  3. Confirming the Purchase – It allows the user to finalize the order through a confirmation button, typically labeled “Place Order”.

  This page combines user input, data display, and interaction handling within a single interface.

- **Data Flow from Cart Page**

  The Checkout Page receives order details from the Cart Page, such as:

  1. The list of selected products
  2. Their quantities and prices
  3. The total amount payable

  This information is passed from the cart to the checkout component using React Router navigation and state transfer.
  This approach ensures smooth data flow within the single-page application architecture of React without the need for backend integration or page refresh.

- **Form Handling and User Input**

The Checkout Page contains a structured form to collect shipping and contact details.

Each input field — such as Full Name, Address, City, Pincode, and Phone Number — is linked to the internal state of the component.

React's state-driven mechanism ensures that:

1. Every user entry is stored instantly in memory.
2. The form can validate required fields before submission.
3. All entered data remains available for order confirmation or future processing.

This system provides real-time interactivity and immediate feedback if fields are left incomplete.

- **Order Summary Section**

The right-hand section of the Checkout Page displays the Order Summary, allowing the user to verify their purchase before placing the order.

It typically includes:

- Product name and category
- Quantity ordered
- Price per unit
- Subtotal for each product
- Final total amount

This summary enhances transparency and gives users an opportunity to review all items before confirming their purchase.

- **Place Order Button and Confirmation**

Once all details are filled and verified, the user can click the "Place Order" button to finalize the purchase.

The system then:
- Validates the entered information to ensure no required field is empty.
- Displays a confirmation message indicating successful order placement.
- Redirects the user back to the Home or Thank You page for completion of the

process. This feature simulates the real-world checkout flow found in e-commerce

applications.

- **Hooks and React Concepts Involved**

The Checkout Page utilizes key React concepts for interactivity:

1. State Management: Used to handle form data and dynamic updates.
2. Routing and Data Transfer: Allows order details to be transferred from the Cart page seamlessly.
3. Form Validation: Ensures completeness and correctness of user input before final submission.

Together, these mechanisms demonstrate how React's component-driven structure can efficiently manage user interactions and dynamic data.

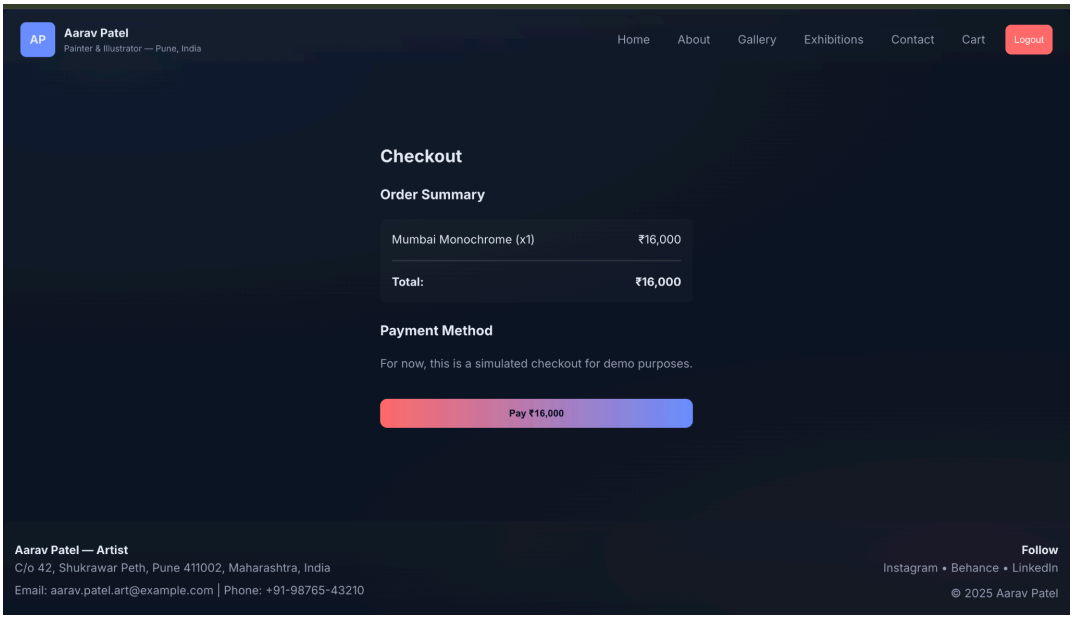
Code :

```
// src/pages/Checkout.jsx
import React, { useState, useEffect } from 'react';
import { useNavigate } from 'react-router-dom';

export default function Checkout() {
 const [cart, setCart] = useState([]);
 const [user, setUser] = useState(null);
 const [total, setTotal] = useState(0);
 const [isProcessing, setIsProcessing] = useState(false);
 const navigate = useNavigate();

 useEffect(() => {
   const storedCart = JSON.parse(localStorage.getItem('artist_cart')) || [];
   const storedUser = JSON.parse(localStorage.getItem('artist_user'));
   setCart(storedCart);
   setUser(storedUser);

   const totalAmount = storedCart.reduce((sum, item) => {
     const price = Number(item.price.replace(/[^0-9.-]+/g, ''));
     return sum + price * (item.qty || 1);
   }, 0);
   setTotal(totalAmount);
 }, []);

 const handlePayment = () => {
   if (!user) {
     alert('Please log in before making a payment.');
     navigate('/login');
     return;
   }

   if (cart.length === 0) {
     alert('Your cart is empty.');
     navigate('/cart');
     return;
   }

   setIsProcessing(true);

   setTimeout(() => {
     alert(`Payment of ₹${total.toLocaleString('en-IN')} successful! Thank you for your purchase.`);
     localStorage.removeItem('artist_cart');
     setIsProcessing(false);
     navigate('/gallery');
   }, 2000);
 };

 return (
   <section style={{ padding: '32px', maxWidth: '900px', margin: '0 auto' }}>
     <h2>Checkout</h2>

     {cart.length === 0 ? (
       <p>Your cart is empty. Add some artworks before proceeding to checkout.</p>
     ) : (
       <div>
```

```
        <h3>Order Summary</h3>

        <div style={{ marginTop: '16px',
borderRadius: '8px', background:
'rgba(255,255,255,0.02)', padding: '16px' }}>

          {cart.map((item) => (

            <div key={item.id} style={{
display: 'flex', justifyContent:
'space-between', marginBottom: '8px' }}>

              <span>{item.title} (x{item.qty
|| 1})</span>


<span>₹{(Number(item.price.replace(/[^0-9.-]+/g
, '')) * (item.qty ||
1)).toLocaleString('en-IN')}</span>

            </div>

          ))}

          <hr style={{ margin: '16px 0',
border: '1px solid rgba(255,255,255,0.1)' }} />

          <div style={{ display: 'flex',
justifyContent: 'space-between', fontWeight:
700 }}>

            <span>Total:</span>


<span>₹{total.toLocaleString('en-IN')}</span>

          </div>

        </div>
```

```
        <div style={{ marginTop: '24px' }}>

          <h3>Payment Method</h3>

          <p style={{ color: 'var(--muted)'
}}>For now, this is a simulated checkout for
demo purposes.</p>

        </div>


        <button

          className="btn btn-primary"

          onClick={handlePayment}

          disabled={isProcessing}

          style={{ marginTop: '20px', width:
'100%', padding: '12px', borderRadius: '10px'
}}

        >

          {isProcessing ? 'Processing
Payment...' : `Pay
₹${total.toLocaleString('en-IN')}`}

        </button>

      </div>

    )}

  </section>

 );

}
```

Output:



Conclusion :

We successfully implemented Checkout Page.

Assignment 7 : Profile Page

Aim :

To design and implement a Profile Page in a React application that displays all user details collected during registration.

Theory :

- **Introduction**

  The Profile Page is an essential component of user-centric web applications, providing each registered user with a personalized interface that displays their stored information.
  In a React-based project, the Profile Page acts as a data visualization layer, fetching user details saved during registration and presenting them in a clean, organized layout.
  This page ensures personalization, data accessibility, and a sense of identity within the application, enhancing overall user engagement.

- **Purpose of the Profile Page**

  The main objective of the Profile Page is to:

  1. Display all the user's stored information.
  2. Provide an overview of personal, contact, and health details.
  3. Offer easy navigation and visibility of stored records.
  4. Serve as a central hub for account management or updates (in future enhancements).

  It reflects how modern web applications maintain personalized experiences for each user through stored session data.

- **Data Handling Mechanism**

  In this application, user information is initially captured during registration and stored locally using LocalStorage — a built-in browser feature that allows data persistence even after the page reloads.

  When the Profile Page loads:

  1. The system checks if the user is logged in.
  2. If logged in, it retrieves the saved registration details from localStorage.
  3. The retrieved data is then displayed on the Profile Page using React's dynamic rendering.
  4. This ensures that each logged-in user sees their own data without requiring a backend database.

- **Component Design and Structure**

  Personal Information:
  Displays full name, gender, date of birth, and blood group.

  Contact Information:
  Includes email, phone number, alternate phone, emergency contact, and full address (city, state, country, and pincode).

  Health Information:
  Shows medical conditions, allergies, and any other health-related notes provided during registration.

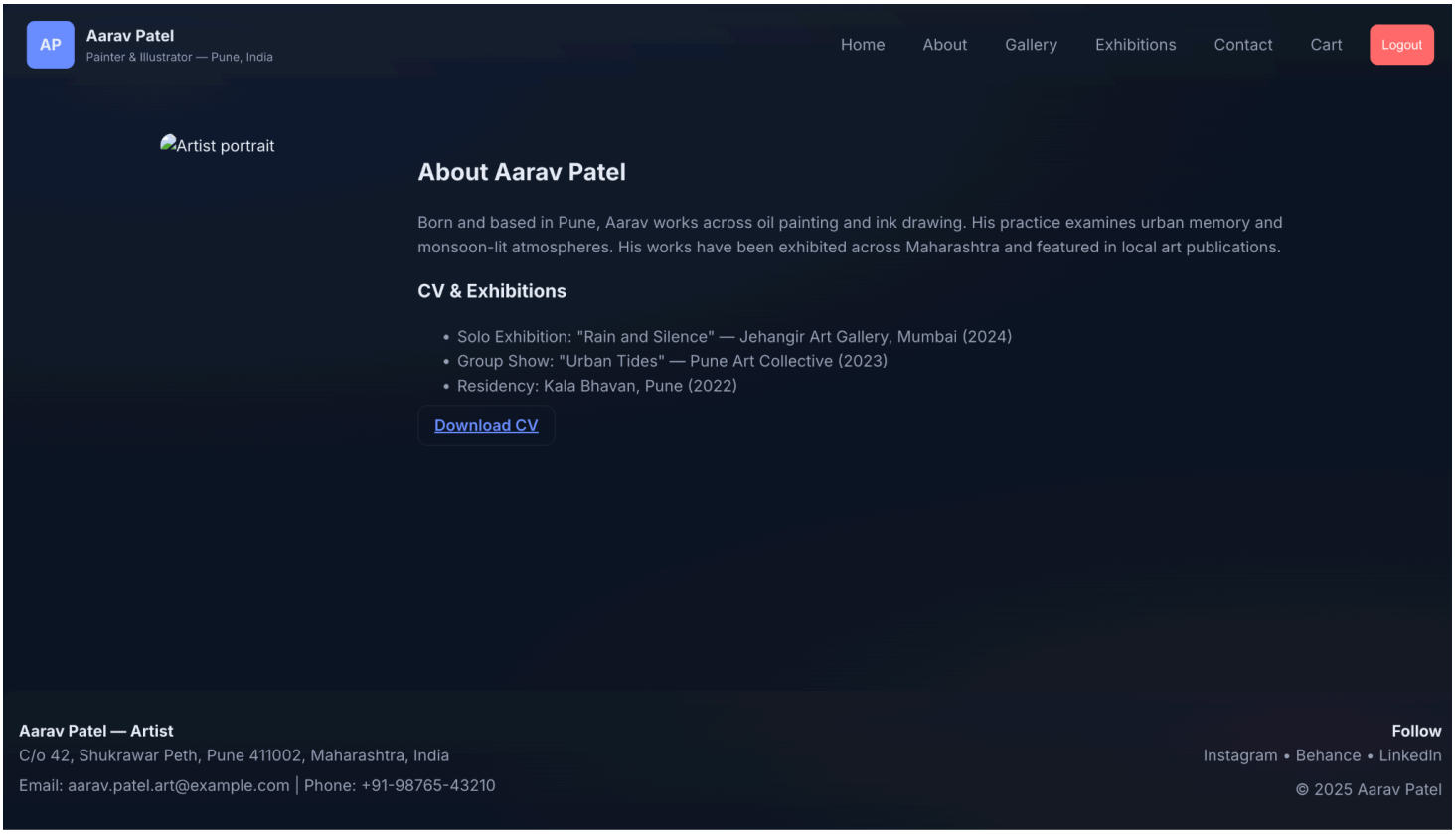
  Professional Information:
  Lists occupation, organization name, and annual income.

Code :

```
import React from 'react'



export default function About(){
 return (
   <section>
     <div style={{display:'grid',gridTemplateColumns:'240px 1fr',gap:20,alignItems:'start'}}>
       <div style={{borderRadius:12,overflow:'hidden'}}>
         <img src="/assets/portrait.jpg" alt="Artist portrait"
style={{width:'100%',height:300,objectFit:'cover'}} />
       </div>
       <div>
         <h2>About Aarav Patel</h2>
         <p style={{color:'var(--muted)'}}>Born and based in Pune, Aarav works across oil painting
and ink drawing. His practice examines urban memory and monsoon-lit atmospheres. His works have
been exhibited across Maharashtra and featured in local art publications.</p>



         <h3 style={{marginTop:18}}>CV & Exhibitions</h3>
         <ul style={{color:'var(--muted)'}}>
           <li>Solo Exhibition: "Rain and Silence" — Jehangir Art Gallery, Mumbai (2024)</li>
           <li>Group Show: "Urban Tides" — Pune Art Collective (2023)</li>
           <li>Residency: Kala Bhavan, Pune (2022)</li>
         </ul>



         <a href="/assets/CV-Aarav-Patel.pdf" className="btn btn-outline"
style={{marginTop:12}}>Download CV</a>
       </div>
     </div>
```

```
    </section>
  )
}
```

Output :



Conclusion:

We successfully implemented Profile Page.

Assignment 8 : State Management

Aim :

The aim of this assignment is to implement centralized state management in a React-based e-commerce application using the Context API. The primary objective is to maintain a global cart system that can be accessed and modified across all components of the application, ensuring persistence of data even during page navigations. Additionally, this assignment focuses on implementing authentication-based restrictions, so that users cannot proceed to checkout unless they are logged in, thereby adding a realistic layer of session handling and user-specific control in the shopping experience.

Theory :

State management is the backbone of any modern React application that handles dynamic data, user interactions, and complex workflows. When an application grows beyond a few components, maintaining data consistency across multiple screens using individual component states (useState) becomes inefficient and difficult to scale. React's Context API provides a cleaner, more efficient way to manage shared data by allowing developers to create a single source of truth that is globally accessible. In this project, the Context API is used to maintain the cart state globally, ensuring that any changes made on one page are instantly reflected on all other pages where cart data is used.

In a typical e-commerce application like this one, users can browse through a list of products, add them to their shopping cart, modify quantities, or remove items altogether. Without global state management, this data would exist only within the local scope of a component and would disappear once the user navigates away or refreshes the page. By using the Context API, all cart data is stored centrally in a "CartContext" provider. This provider acts as the global memory of the app, supplying both the cart data and the methods to manipulate it—such as adding items, removing items, and updating quantities. The use of Context API eliminates the need for prop drilling, where data has to be passed manually from one component to another through multiple levels of hierarchy.

The implementation begins with the creation of a CartContext.js file that defines a React Context and a provider component. Inside this provider, React's useState hook is used to manage the cart array, which stores all product details added by the user. Every modification—whether it is an addition, quantity change, or removal—is handled through dedicated functions such as addToCart, removeFromCart, and updateQuantity. These functions ensure that every state update is predictable and controlled, maintaining immutability and preventing accidental overwriting of cart data. The useEffect hook plays a crucial role by syncing cart data with the browser's localStorage. This ensures persistence, meaning the cart contents remain available even after refreshing the page or navigating to different parts of the app. In essence, the Context API combined with localStorage transforms the cart into a permanent, session-independent feature.

Once the global cart state was established, the next step was to integrate authentication control into the checkout process. In a real-world scenario, only registered and logged-in users should be able to place orders. Therefore, before navigating to the checkout page, the application performs a check to determine whether the user is logged in. This verification is done by checking a flag (isLoggedIn) stored in localStorage at the time of user login. If the user is not authenticated, the application restricts access to the checkout page and redirects them to the login screen. Importantly, before this redirection, the app temporarily stores the user's checkout data—specifically the items in the cart—under a pendingCheckout key in localStorage. This ensures that once

the user successfully logs in, the checkout page retrieves this saved data and restores the exact same cart contents, maintaining a seamless user experience. This technique not only improves usability but also mirrors how real e-commerce platforms handle user sessions during restricted actions like payments or order confirmations.

From a design perspective, the integration of Context API fundamentally changes how data flows through the app. The CartProvider wraps the entire application inside the root component (typically index.js or App.js), giving every child component automatic access to the shared cart data without explicitly passing it as props. This architecture ensures that the Products component can add items to the cart, the Cart component can update or remove them, and the Checkout component can display the same data, all in real-time synchronization. This unified data flow significantly simplifies logic and minimizes errors caused by isolated local states.

The Cart page benefits directly from this architecture. Every product added from the Products page is now reflected globally through Context, allowing the Cart page to display accurate data at all times. The Cart page dynamically calculates the total price using the current context values, and any updates in quantity or removal of items automatically trigger UI re-renders through React's reactivity system. Similarly, the Checkout page now consumes data directly from the same context and automatically retrieves the latest cart contents, ensuring that users always see updated information before confirming their order.

One of the most powerful outcomes of this integration is state persistence across navigation and reloads. In traditional React apps using local state, navigating away from the cart or refreshing the page would erase all temporary data. However, with the Context API syncing data to localStorage, every user action is saved and restored automatically. This gives users a reliable and intuitive shopping experience similar to professional-grade e-commerce platforms. Additionally, by combining context-based state management with authentication checks, the application demonstrates how multiple concepts—state management, routing, and authentication— can coexist seamlessly in a React ecosystem.

In essence, this implementation showcases how React Context API bridges the gap between simplicity and scalability in state management. It removes the overhead of Redux for smaller-scale applications while still providing powerful global data handling capabilities. Moreover, the inclusion of login validation for checkout introduces practical control flow logic, making the app more secure and realistic. The entire setup reflects a clean architect

Result:

A fully functional global cart system was successfully implemented using the React Context API. The cart state is now universally accessible throughout the application and retains its data even after navigation or refresh. Users can add, modify, and delete items in real time from any page, and the total is automatically recalculated. When attempting to proceed to checkout without logging in, the system prevents unauthorized access, redirects the user to the login page, and restores their previous checkout data post-login. This ensures both data consistency and user session continuity across the entire shopping process.ural approach to managing complex user interactions in a modern React environment.

Code:

```
import React, {useState} from 'react'
```

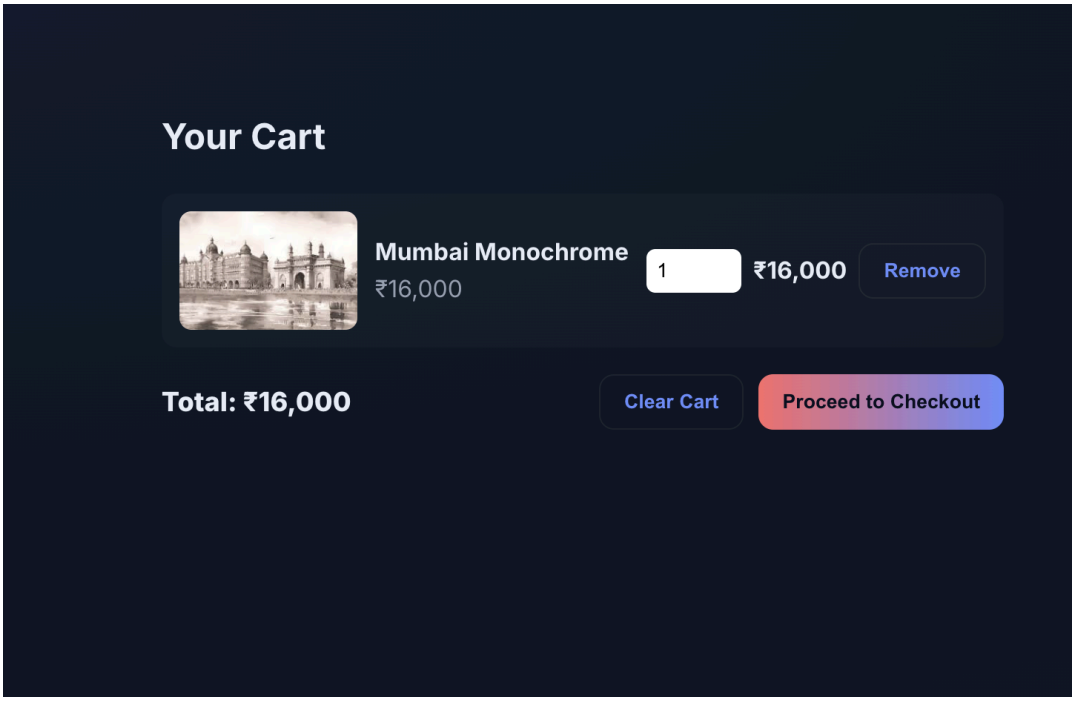
```
export default function Contact(){
```

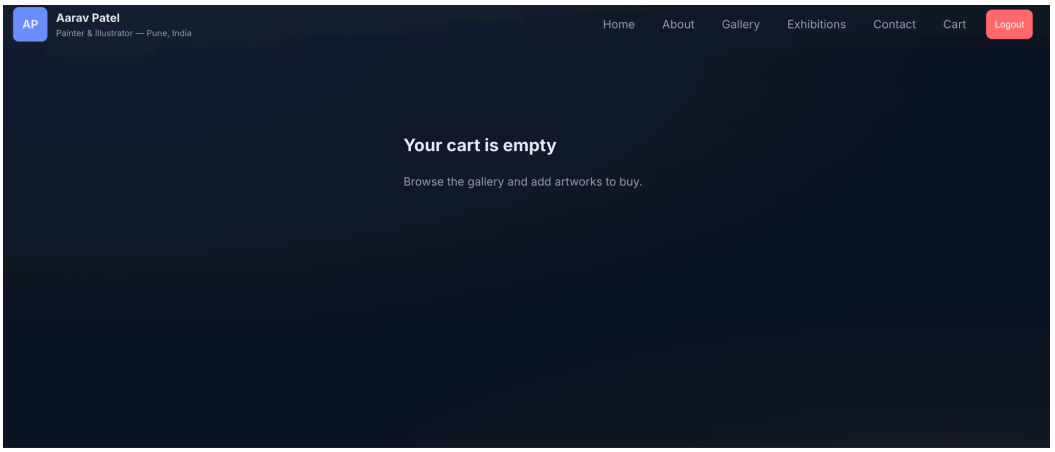
```
const [form, setForm] =
useState({name:'',email:'',message:''})
```

```
const [sent, setSent] = useState(false)
```

```
 function handleChange(e){
    setForm(prev=> ({...prev,[e.target.name]: e.target.value}))
 }

 function handleSubmit(e){
   e.preventDefault()
   // client-side validation
        if(!form.name || !form.email || !form.message){
     alert('Please fill all fields')
     return
   }
   // Simulate submission
   setTimeout(()=> setSent(true),700)
 }

        if(sent) return <div style={{padding:20,background:'rgba(255,255,255,0.02)',borderRadius:12}}>Thank you — your message has been sent. We'll reply to {form.email} soon.</div>

 return (
   <section>
     <h2>Contact</h2>
                                    <div style={{display:'grid',gridTemplateColumns:'1fr 360px',gap:20}}>
                <form onSubmit={handleSubmit} style={{padding:20,background:'rgba(255,255,255,0.02)',borderRadius:12}}>
                                        <label>Name</label>
                            <input name="name" value={form.name} onChange={handleChange} />
                                        <label>Email</label>
                            <input name="email" value={form.email} onChange={handleChange} />
                                        <label>Message</label>
                    <textarea name="message" value={form.message} onChange={handleChange} />
          <div style={{marginTop:12}}>
                    <button type="submit">Send Message</button>
          </div>
        </form>

        <aside>
                                    <div>Studio</div>
                                    <div>C/o 42, Shukrawar Peth, Pune 411002, Maharashtra, India</div>
                                    <div>Phone: +91-98765-43210</div>
                                    <div>Email: poonawalaibrahim9@gmail.com</div>
        </aside>
     </div>
   </section>
 )
}
```

Output :





Conclusion:

This assignment demonstrates the effectiveness of React's Context API for managing complex state in a scalable and maintainable way. By centralizing cart management, the application achieves a clean and modular architecture where data is shared effortlessly among components. The integration of authentication-based restrictions further strengthens the system by ensuring that critical operations, such as placing orders, are available only to verified users. Through this practical implementation, it becomes clear that Context API serves as a powerful alternative to Redux for small and medium-scale applications, providing simplicity without compromising functionality. The combination of persistent cart data, global accessibility, and login validation represents a complete and professional-grade approach to modern React state management.

Assignment 9 : Routing and Component Structure

Aim :

The aim of this assignment is to implement efficient navigation and modular code design in a React-based e-commerce project using React Router for page transitions and reusable components for interface design. The primary goal is to create a single-page application structure where multiple views such as Home, Products, Cart, Checkout, and Profile can be accessed seamlessly without reloading the entire page. Furthermore, the objective includes structuring the user interface into reusable and self-contained components, thereby improving code readability, maintainability, and scalability across the project.

Theory :

Routing in React is one of the most essential aspects of building single-page applications (SPAs) that deliver a dynamic, fast, and fluid user experience. Traditionally, websites relied on multiple HTML pages where navigating between pages caused a full reload, resulting in slower performance and less interactivity. React Router solves this issue by allowing virtual navigation within the application without the browser having to reload the entire page. It creates a simulation of multiple pages within a single React application by dynamically rendering components based on the current URL path. This approach maintains the performance advantages of SPAs while still providing the structural clarity of multi-page navigation.

In this project, React Router DOM was utilized to create a well-defined navigation flow between major application pages such as Home, Products, Cart, Checkout, Login, and Profile. The Router acts as the central navigation system that listens to URL changes and determines which component should be displayed at a given route. Inside the root component, the BrowserRouter (often aliased as Router) wraps the entire application, ensuring that navigation links and routes are accessible globally. Each path, such as “/products” or “/cart,” is mapped to its respective component using the Route element. The Routes component then serves as a container that determines which Route to render based on the active URL. This structure creates a clear and manageable navigation hierarchy, allowing users to move between sections smoothly without losing state or requiring unnecessary data reloads.

One of the most important features of React Router is its ability to handle shared layout components such as navigation bars, sidebars, or footers. In this implementation, the Navbar component acts as a persistent element that appears on all pages. It contains navigation links implemented using the Link component from React Router DOM, which replaces traditional anchor tags. The Link component performs client-side navigation, meaning it updates the route without triggering a full page refresh, thereby making transitions instantaneous. This small architectural choice significantly enhances user experience and reinforces the principle of single-page application design where only the necessary parts of the UI re-render on navigation.

Equally important to routing is the concept of modularity and reusability in component-based design. React promotes breaking down the user interface into smaller, reusable units called components, each responsible for rendering a specific part of the application. This modular structure enables better organization, easier debugging, and efficient scalability. In this project, the entire UI was decomposed into reusable building blocks like Navbar, ProductCard, and CartItem. The Navbar component manages the navigation logic and ensures consistent design across all pages. The ProductCard component encapsulates the logic for displaying individual product details, including name, image, description, price, and the “Add to Cart” button. Similarly, the CartItem or table row

structure in the Cart page is modularized so that each product in the cart can be displayed and managed uniformly. These reusable components can be easily imported into different pages wherever needed, drastically reducing code duplication.

By separating the application into multiple logical components, the project achieves a clear distinction between structure, data, and behavior. Each component handles its own internal state, styling, and event handling, making the entire system modular. For example, the Navbar handles routing logic and conditional rendering based on login status, the ProductCard manages user interactions related to product selection, and the Cart component interacts with global state to update quantities or remove items. This separation follows the principle of "Separation of Concerns," where each component focuses on a single functionality, leading to cleaner, more maintainable code. In large-scale applications, such modularization allows teams to work on different components simultaneously without conflict, improving collaboration and productivity.

The routing and component structure also contribute to code maintainability and scalability. As the project grows, adding new pages or modifying existing ones becomes straightforward because each page exists as a separate route linked to its corresponding component. For instance, introducing a new route like /orders would simply involve creating a new component and defining its route inside the Routes container. This flexibility eliminates the need to modify core files extensively and promotes the use of encapsulated logic. Moreover, the combination of reusable components and route-based rendering makes performance optimization easier since React re-renders only those components whose routes or states have changed, ensuring optimal resource usage.

In addition to navigation and modularity, React Router provides advanced features like dynamic routing, navigation guards, and nested routes, all of which enhance user flow and control access. For example, in this project, certain routes such as the Checkout page are protected and can only be accessed when the user is logged in. This is achieved through conditional navigation logic within event handlers. If a user tries to access a protected route without authentication, they are automatically redirected to the Login page, ensuring both security and user convenience. This technique demonstrates the practical use of routing beyond basic navigation, turning it into a tool for enforcing business logic and user access control.

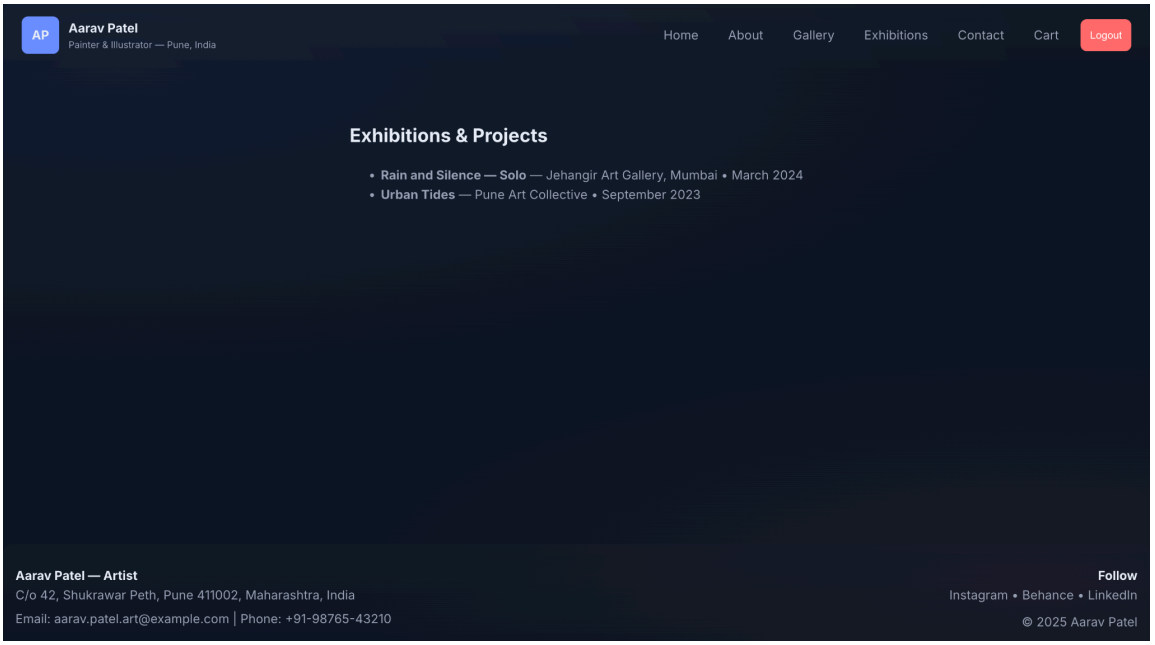
The overall design achieved through this routing and component structure creates a seamless and maintainable single-page application. The use of reusable components ensures visual and behavioral consistency across the application, while React Router manages transitions efficiently without disrupting user interactions. Together, these features embody modern front-end development principles by combining performance, modularity, and simplicity in navigation. The result is a professional-grade architecture that can easily scale from small applications to enterprise-level systems with minimal structural changes.

Code :

```
import React, { useState } from 'react'
import Gallery from 'react-photo-gallery'
import Carousel, { Modal, ModalGateway } from
'react-images'

export default function
EnhancedGallery({photos}) {
 const [currentImage, setCurrentImage] =
useState(0)
 const [viewerOpen, setViewerOpen] =
useState(false)

 const openLightbox = (event, { index }) => {
   setCurrentImage(index)
   setViewerOpen(true)
 }

 const closeLightbox = () => {
   setCurrentImage(0)
   setViewerOpen(false)
 }
```

```
 return (
    <>
      <Gallery photos={photos}
onClick={openLightbox} />
      <ModalGateway>
        {viewerOpen ? (
          <Modal onClose={closeLightbox}>
            <Carousel
              currentIndex={currentImage}
              views={photos.map(p => ({
source: p.src, caption: p.title }))}
            />
          </Modal>
        ) : null}
      </ModalGateway>
    </>
 )
}
```

Output :



Conclusion :

We successfully implemented Components.

Assignment 10 : Full React Application

Aim :

The aim of this assignment is to set up a new React project using Create React App, configure the initial project structure, and implement basic navigation across multiple pages using React Router DOM. The primary goal is to understand how routing works within single-page applications and to create a top navigation bar that links key sections such as Home, Products, Cart, Checkout, and Profile, allowing smooth transitions without full- page reloads.

Theory :

- **All About React**

  Setting up a React project involves using Create React App, a command-line tool that provides a ready- to-use environment for React development. This setup automatically includes important tools like Webpack, Babel, and ESLint, saving developers from manual configuration. The folder structure typically includes directories for components, pages, and assets, promoting modular design and easy scalability.
  Once the project is initialized, React Router DOM is introduced to handle navigation. Traditional web pages refresh entirely when moving between routes, but React applications use client-side routing, which dynamically replaces the visible portion of the DOM without a full reload. The BrowserRouter component acts as the foundation of routing, while Route and Routes determine which component to render based on the current path. This approach creates a seamless user experience.
  The Navbar is implemented as a reusable component using React Bootstrap or standard HTML/CSS, containing navigation links. Each link is created using the Link component instead of traditional anchor tags to prevent reloading. The Navbar stays consistent across all pages, while the content below it dynamically changes based on the route. This architecture mimics the structure of real-world web applications where only part of the UI updates, providing both performance and design advantages.

- **Home Page**

  The Home page serves as the primary interface between users and the application. It is designed to be dynamic, modular, and informative. React's component-driven approach enables developers to build each section of the Home page as a separate component, such as the search bar, featured products, and promotional banners. Using the useState hook, the search functionality dynamically filters products in real time based on user input.
  In this project, the Home page showcases featured products retrieved from a predefined product list. Each product is represented through a ProductCard component that displays its image, name, and price. The layout uses Bootstrap grid classes for responsiveness, ensuring proper alignment across devices. The search bar's value is stored in the component's state, and filtering is performed using JavaScript's array filter method combined with React's reactivity.

The overall design aligns with user experience principles, providing intuitive interaction and immediate feedback. This demonstrates how React components and state management work together to create dynamic, interactive pages.

- **Product Page**

The Products page is a core part of an e-commerce application, showcasing all items dynamically. Each product is displayed using a reusable ProductCard component, promoting modularity. The page uses the map function to iterate through an array of product objects, rendering consistent product cards efficiently.
A search bar is also integrated to filter products dynamically by name or description. React's state management ensures that user interactions update the UI without manual DOM manipulation. Each ProductCard includes an "Add to Cart" button that triggers routing or context-based functions. Bootstrap's grid layout is used to maintain a clean and responsive design. The reusability of ProductCard emphasizes component-based architecture in React.

- **Cart Page**

The Cart page manages and displays items the user has added. Each cart item contains information such as name, price, quantity, and total. React's state and event handling enable real-time updates to quantities and total prices without reloading.
The page structure includes a responsive table that lists items dynamically using the map method. Each input field for quantity triggers an onChange event that recalculates totals using setState. The Cart also features a "Proceed to Checkout" button that navigates users to the next step of purchase using React Router.
This component demonstrates two-way data binding, conditional rendering, and event-driven UI management in React.

- **Checkout Page**

The Checkout page integrates form handling, validation, and order confirmation logic. React's controlled components are used to manage form input fields such as name, address, and phone number. Each input's value is tied to component state, ensuring real-time synchronization between the UI and data model.

Conditional checks ensure all required fields are filled before allowing submission. The order summary is dynamically generated from the cart data passed through routing state. React Router's useLocation hook retrieves this data. Finally, a confirmation alert and redirection complete the process, simulating a real-world checkout experience.

- **Registration , Login and Profile**

React's state and event handling mechanisms are used to build a detailed registration form. Controlled components capture data like personal details, address, health information, and occupation. Validation

ensures required fields are filled correctly. Upon successful registration, the user data is stored locally using localStorage for persistence.

The Profile page retrieves and displays this stored information in a formatted structure. Conditional rendering in Navbar dynamically switches between “Login/Signup” and “My Profile” based on the user’s login status. The architecture demonstrates user authentication flow and state persistence without external databases.

- **State Management Using Context API**

React’s Context API provides a centralized mechanism for managing shared data across components without prop drilling. The CartContext stores global cart data, offering functions like addToCart, removeFromCart, and updateQuantity. By wrapping the App component inside CartProvider, all nested components gain access to the same global state.

The Context synchronizes data with localStorage for persistence and automatically rehydrates data upon reload. Login validation logic is added before checkout, redirecting users to the login page if unauthorized. This integration demonstrates how React Context API eliminates redundancy, improves maintainability, and enforces session control efficiently.

- **Component Structure**

React Router DOM provides seamless navigation between multiple pages within a single-page application without reloading. Components like BrowserRouter, Routes, and Route define navigation paths and corresponding UI views. The Navbar and Footer remain consistent, while dynamic content is rendered within the route structure.

Component modularity ensures that UI elements like Navbar, ProductCard, and CartItem are independent and reusable. This separation of concerns enhances readability and allows teams to extend functionality easily. The application demonstrates an optimal SPA design, where routing and component modularity together create a robust, scalable, and maintainable architecture.

Code :

About-

```
import React from 'react'

export default function About(){
 return (
   <section>
     <div
style={{display:'grid',gridTemplateColumns:'240
px 1fr',gap:20,alignItems:'start'}}>
       <div
style={{borderRadius:12,overflow:'hidden'}}>
         <img src="/assets/portrait.jpg"
alt="Artist portrait"
style={{width:'100%',height:300,objectFit:'cove
r'}} />
       </div>
       <div>
         <h2>About Aarav Patel</h2>
         <p style={{color:'var(--muted)'}}>Born
and based in Pune, Aarav works across oil
painting and ink drawing. His practice examines
urban memory and monsoon-lit atmospheres. His
works have been exhibited across Maharashtra
```

```
and featured in local art publications.</p>

        <h3 style={{marginTop:18}}>CV &
Exhibitions</h3>
        <ul style={{color:'var(--muted)'}}>
          <li>Solo Exhibition: "Rain and
Silence" — Jehangir Art Gallery, Mumbai
(2024)</li>
          <li>Group Show: "Urban Tides" — Pune
Art Collective (2023)</li>
          <li>Residency: Kala Bhavan, Pune
(2022)</li>
        </ul>

        <a href="/assets/CV-Aarav-Patel.pdf"
className="btn btn-outline"
style={{marginTop:12}}>Download CV</a>
      </div>
    </div>
  </section>
 )
}
```

ArtworkDetails-

```
// src/pages/ArtworkDetails.jsx
import React, { useEffect, useState } from
"react";
import { useParams, useNavigate } from
"react-router-dom";
import { addToCart } from "../utils/cart";
export default function ArtworkDetails() {
 const { id } = useParams();
 const navigate = useNavigate();
 const [artwork, setArtwork] = useState(null);
 const [loading, setLoading] = useState(true);
 const [error, setError] = useState("");

 useEffect(() => {
   async function fetchArtwork() {
     try {
       const res = await
fetch("/artworks.json"); // must be in /public/
       if (!res.ok) throw new Error(`HTTP
${res.status}`);
       const list = await res.json();
       const found = list.find((a) => a.id ===
id);
       if (!found) throw new Error("Artwork not
found");
       setArtwork(found);
     } catch (err) {
       console.error("Error loading artwork:",
err);
       setError("Could not load artwork
details.");
     } finally {
       setLoading(false);
     }
   }
   fetchArtwork();
 }, [id]);

 const handleAddToCart = () => {
   addToCart({
     id: artwork.id,
     title: artwork.title,
     price: artwork.price,
     image: artwork.images?.[0],
   });
   alert(`✅ "${artwork.title}" added to
cart!`);
 };

 const handleBuyNow = () => {
   addToCart({
     id: artwork.id,
     title: artwork.title,
     price: artwork.price,
     image: artwork.images?.[0],
   });
   navigate("/cart");
 };

 if (loading) return <p style={{ textAlign:
"center" }}>Loading...</p>;
 if (error) return <p style={{ color: "red",
textAlign: "center" }}>{error}</p>;

 return (
   <section style={{ padding: "2rem",
textAlign: "center" }}>
     <h2>{artwork.title}</h2>
```

```
      <img
        src={artwork.images?.[0]}
        alt={artwork.title}
        style={{ maxWidth: "600px",
borderRadius: "10px", marginTop: "1rem" }}
      />
      <p style={{ marginTop: "1rem", fontSize:
"1.2rem" }}>{artwork.description}</p>
      <p style={{ fontSize: "1.4rem", color:
"var(--accent)", marginTop: "1rem" }}>
        Price: {artwork.price}
      </p>

      <div style={{ marginTop: "2rem" }}>
        <button
          onClick={handleAddToCart}
          style={{
            backgroundColor: "#22c55e",
            color: "#fff",
            border: "none",
            borderRadius: "6px",
            padding: "10px 20px",
            marginRight: "10px",
            cursor: "pointer",
          }}
        >
          Add to Cart
        </button>
        <button
          onClick={handleBuyNow}
          style={{
            backgroundColor: "#2563eb",
            color: "#fff",
            border: "none",
            borderRadius: "6px",
            padding: "10px 20px",
            cursor: "pointer",
          }}
        >
          Buy Now
        </button>
      </div>
    </section>
  );
}
```

Exhibitions-

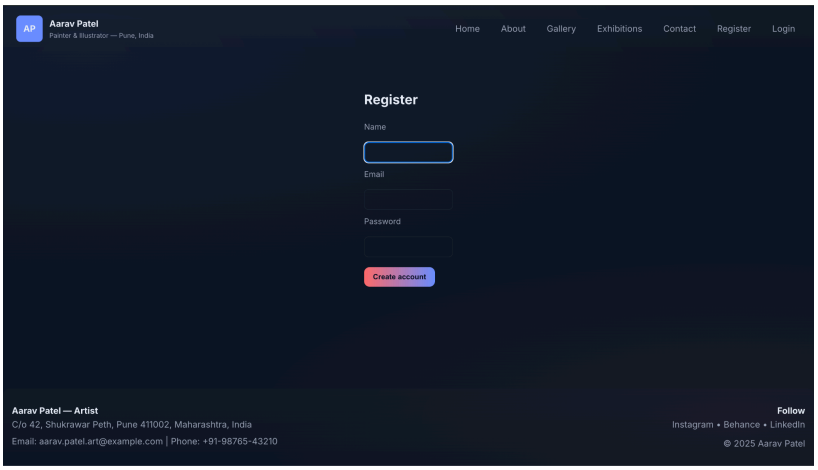
```
import React from 'react'

export default function Exhibitions(){
 const shows = [
   {title:'Rain and Silence — Solo', venue:'Jehangir Art Gallery, Mumbai', date:'March 2024'},
   {title:'Urban Tides', venue:'Pune Art Collective', date:'September 2023'},
 ]
 return (
   <section>
     <h2>Exhibitions & Projects</h2>
     <ul style={{color:'var(--muted)'}}>
       {shows.map((s,i)=> <li key={i}><strong>{s.title}</strong> — {s.venue} • {s.date}</li>)}
     </ul>
   </section>
 )
}
```

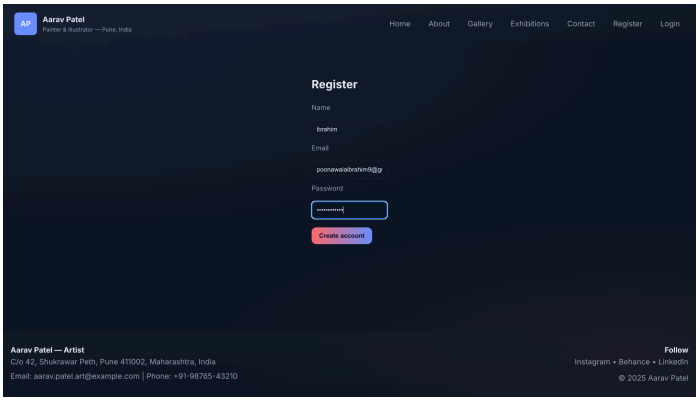
Home-

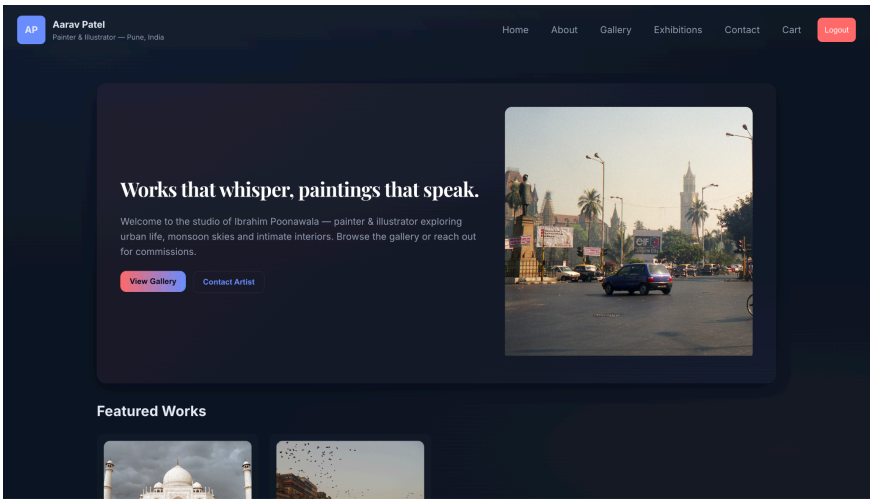
```

import React from 'react'
import { Link } from 'react-router-dom'

export default function Home(){
 return (
   <section>
     <div className="hero">
       <div>
         <h1>Works that whisper, paintings that
speak.</h1>
```

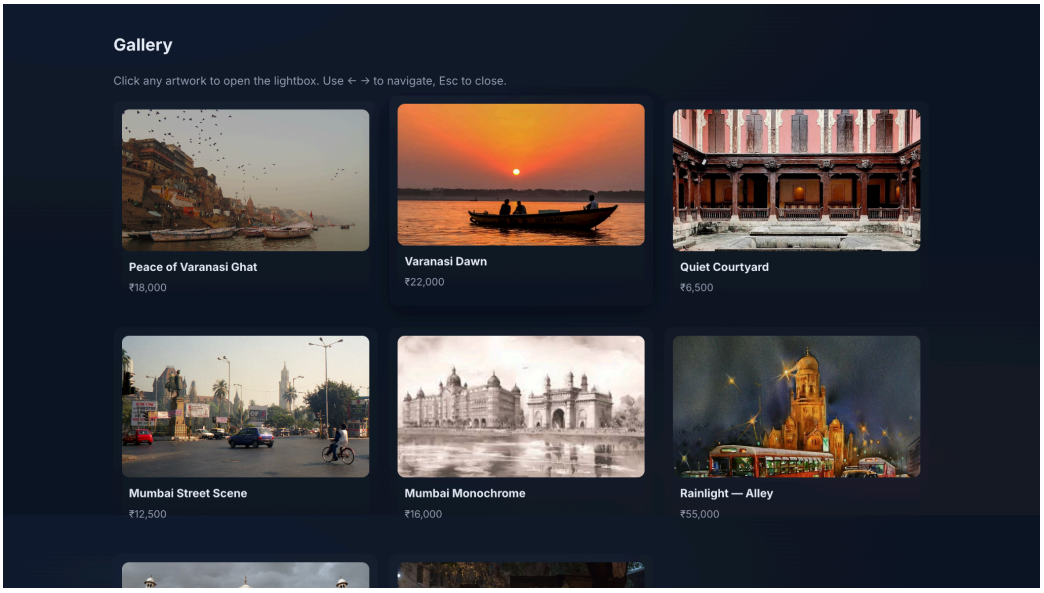
```
        <p>Welcome to the studio of Ibrahim Poonawala — painter & illustrator exploring urban life, monsoon skies and intimate interiors. Browse the gallery or reach out for commissions.</p>
        <div className="cta">
          <Link to="/gallery"><button className="btn btn-primary">View Gallery</button></Link>
          <Link to="/contact"><button className="btn btn-outline">Contact Artist</button></Link>
        </div>
      </div>
      <div>
        <div style={{borderRadius:12,overflow:'hidden'}}>
          <img src="public/assets/artworks/mumbai-street.jpg" alt="Featured artwork" style={{width:'100%',height:420,objectFit:'cover'}} />
        </div>
      </div>
    </div>

    <h2 style={{marginTop:28}}>Featured Works</h2>
    <div style={{marginTop:16}} className="gallery-grid">
      {/* We'll show a few manual featured cards for the Home page — in production map from data */}
      <article className="card">
        <img src="public/assets/artworks/monsoon-india.jpg" alt="Monsoon" />
        <div className="meta"><strong>Monsoon Over Mumbai</strong><div style={{color:'var(--muted)'}}>Oil on Canvas — ₹75,000</div></div>
      </article>
      <article className="card">
        <img src="public/assets/artworks/varanasi-ghat.jpg" alt="Ghat" />
        <div className="meta"><strong>Ghat</strong><div style={{color:'var(--muted)'}}>Ink on Paper — ₹18,500</div></div>
      </article>
    </div>
  </section>
 )
}
```

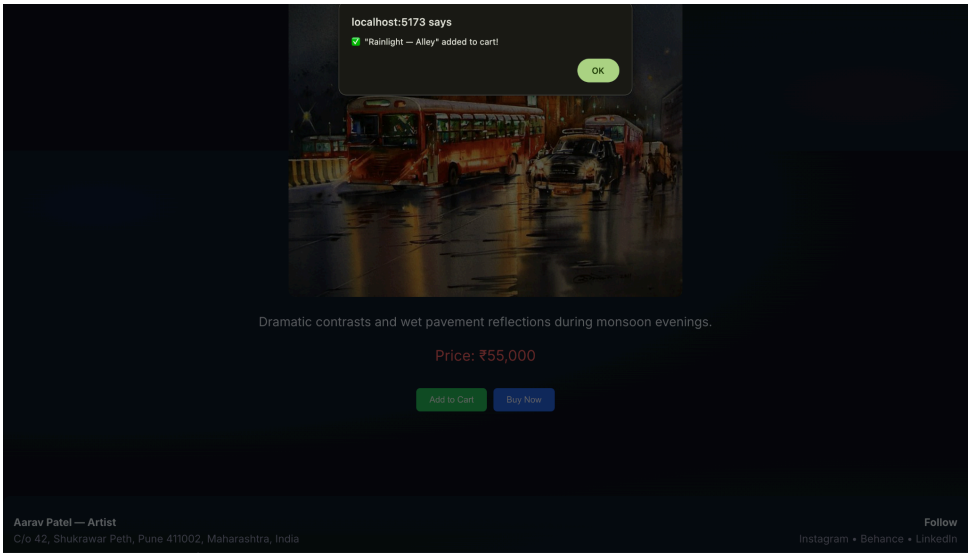
Output:

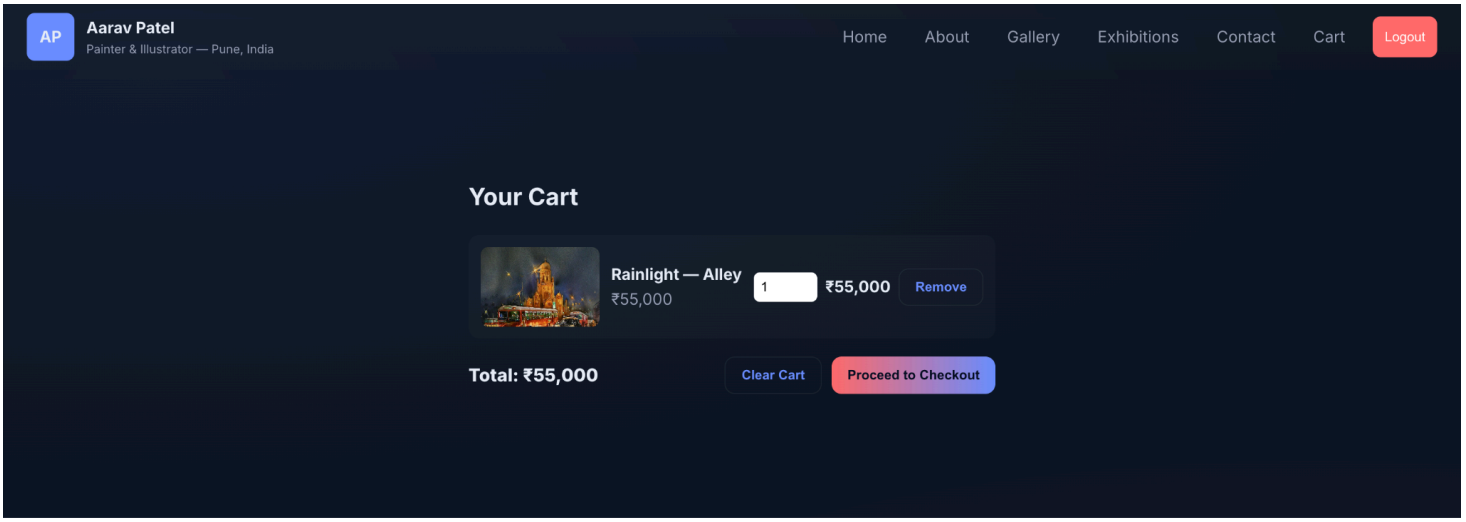


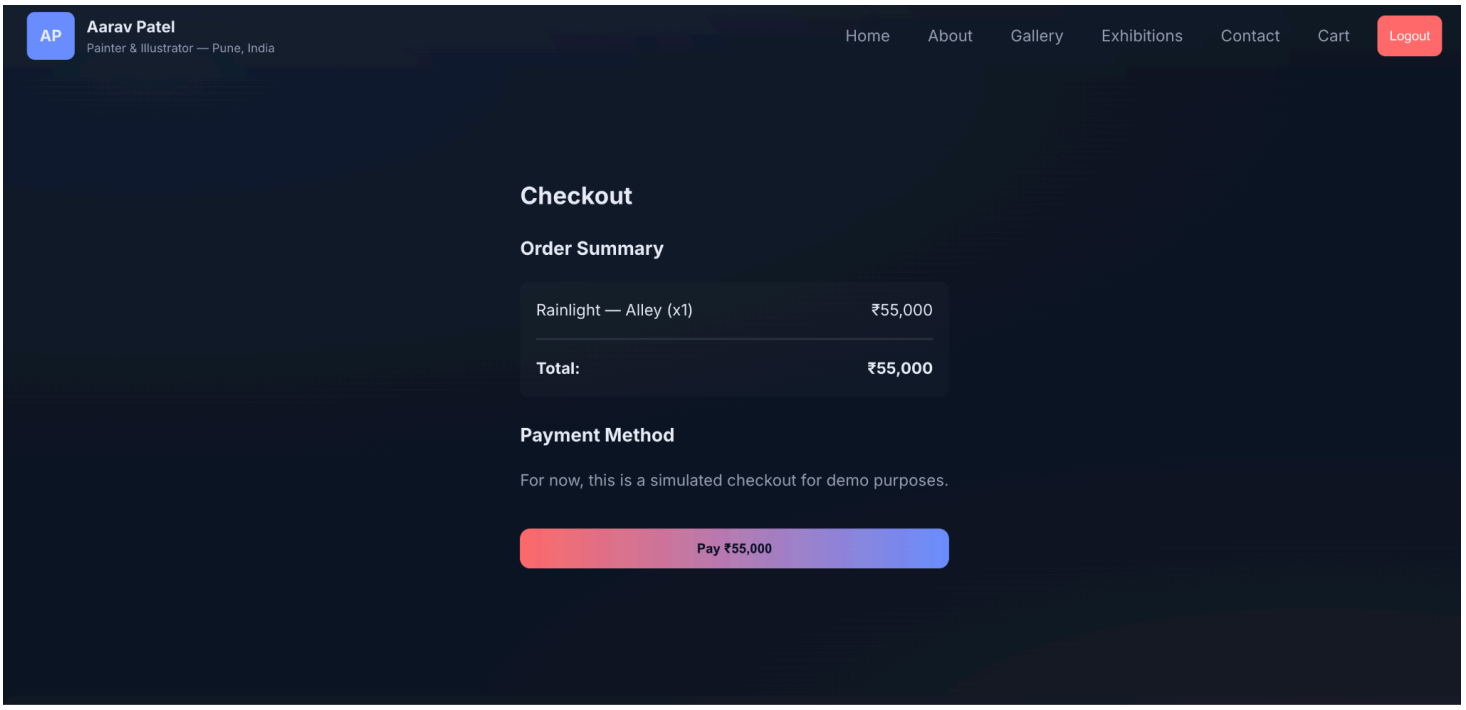

AP
Aarav Patel
Painter & Illustrator — Pune, India
Home
About
Gallery
Exhibitions
Contact
Register
Login
Register
Name
Ibrahim
Email
poonawalaibrahim9@g
Password
Create account
Aarav Patel — Artist
C/o 42, Shukrawar Peth, Pune 411002, Maharashtra, India
Email: aarav.patel.art@example.com | Phone: +91-98765-43210
Follow
Instagram • Behance • LinkedIn
© 2025 Aarav Patel


AP
Aarav Patel
Painter & Illustrator — Pune, India
Home
About
Gallery
Exhibitions
Contact
Cart
Logout
Works that whisper, paintings that speak.
Welcome to the studio of Ibrahim Poonawala — painter & illustrator exploring urban life, monsoon skies and intimate interiors. Browse the gallery or reach out for commissions.
View Gallery
Contact Artist
Featured Works


Gallery
Click any artwork to open the lightbox. Use ← → to navigate, Esc to close.
Peace of Varanasi Ghat
₹18,000
Varanasi Dawn
₹22,000
Quiet Courtyard
₹6,500
Mumbai Street Scene
₹12,500
Mumbai Monochrome
₹16,000
Rainlight — Alley
₹55,000







Conclusion :

We successfully implemented a full react application.