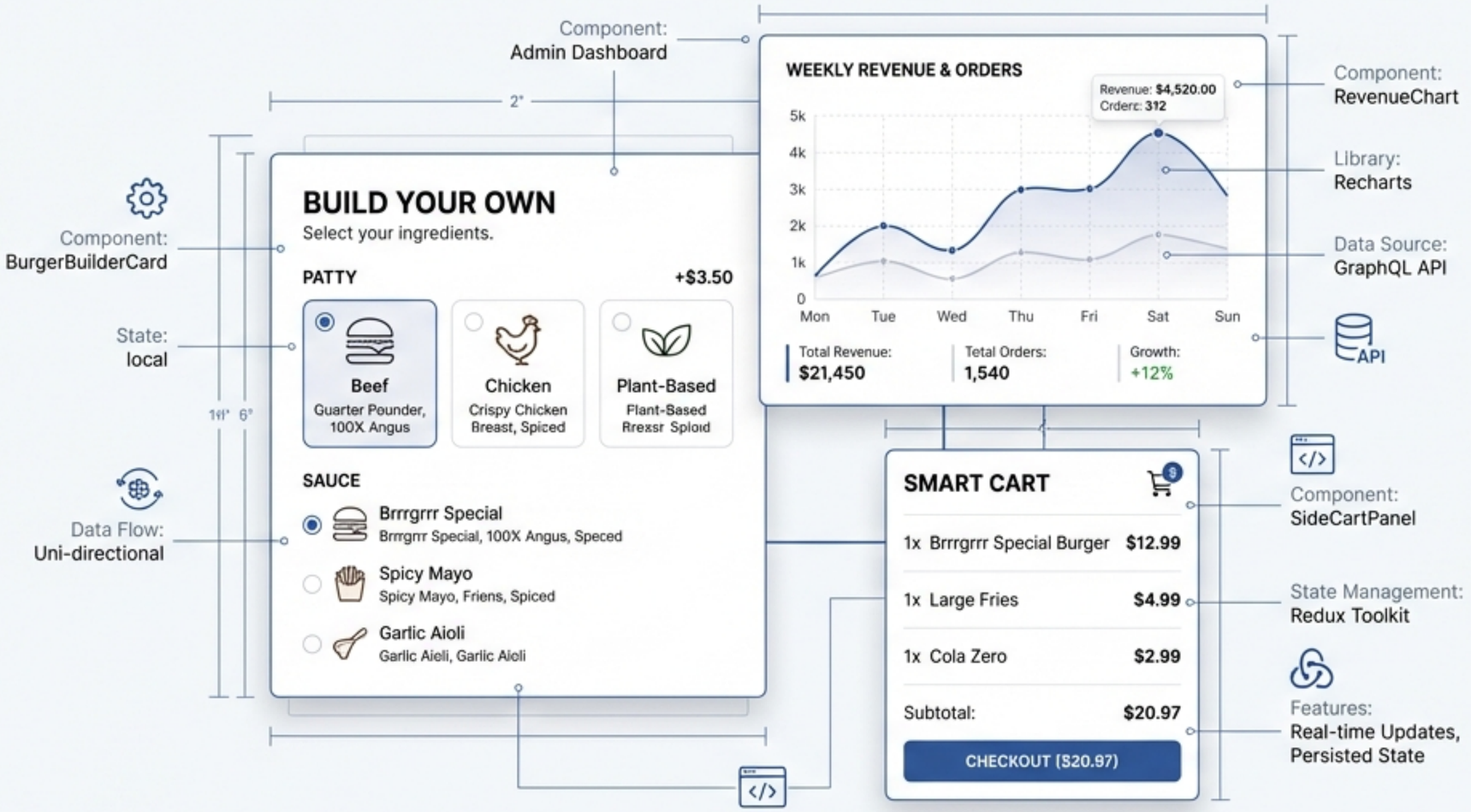
A Deliberate Demonstration of Modern Frontend Engineering

A Technical Deep-Dive into the 'Brrrgrrr' React Application



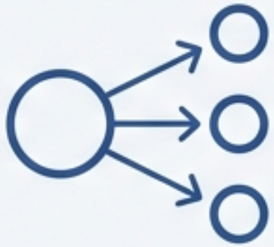
Developed as a Frontend Lab Project demonstrating proficiency in React and Modern JavaScript.

The Core Objectives

This project was built to master and demonstrate proficiency in four mandatory frontend concepts. The following slides showcase how these concepts were implemented to create a dynamic, high-performance application.

Arrays & Data Structures

Utilise arrays for all core data layers, from menus and ingredients to dynamic cart management.

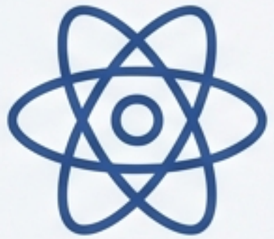


Higher-Order Functions

Implement `map`, `filter`, and `reduce` for clean, efficient data processing and transformation.

ES6+ Compatibility

Employ modern JavaScript syntax including Modules, Arrow Functions, and Destructuring for a clean, component-based architecture.



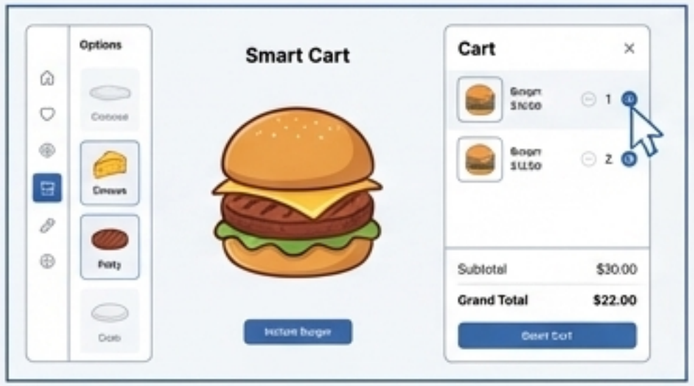
DOM Manipulation via React

Leverage React's Virtual DOM for dynamic rendering, conditional UI, and high-performance updates.

The Result: A Feature-Rich Burger Ordering Platform

Complete customisation from scratch.

Dynamic filtering and search.



Real-time updates and calculations.

Visual mock data analytics.

A pure frontend application built with React, emphasising interactive UI design and efficient state management.

Proof Point 1: Efficient Data Transformation with HOFs

The application leverages the power of ES6 Higher-Order Functions (`map`, `filter`, `reduce`) for clean, declarative, and efficient data transformation, avoiding complex loops and manual state iteration.

This approach is fundamental to powering key interactive features:

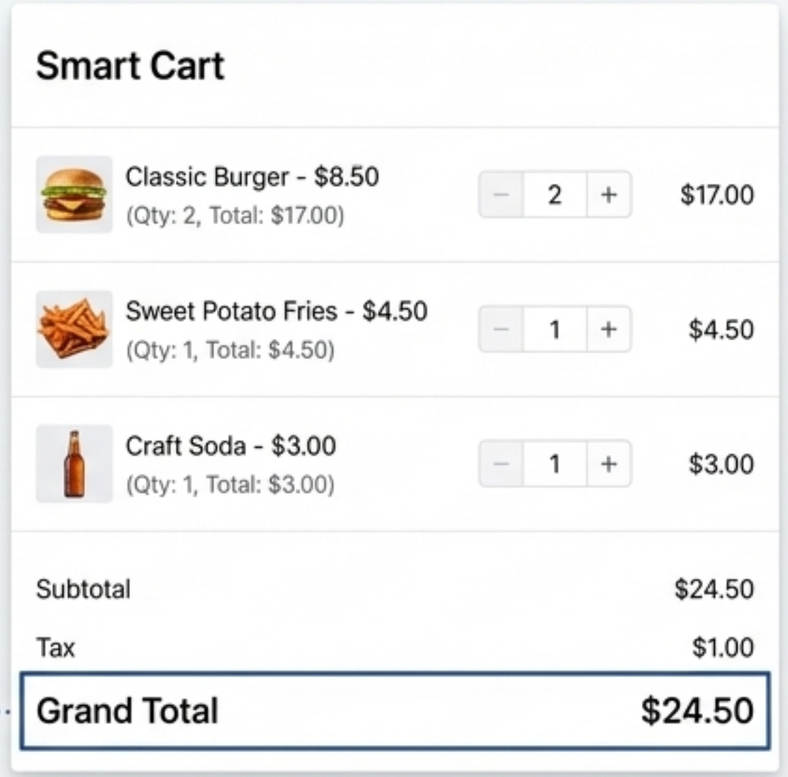
* The **Smart Cart**'s dynamic price calculation.
* The **Interactive Menu**'s filtering system.
* The rendering of all dynamic lists, from menu items to notifications.

HOFs in Action: Calculating the Cart Total with `reduce()`

`CartContext.js" in Source Sans Pro

```
// Example snippet from CartContext
const calculateTotal = (items) => {
  return items.reduce((total, item) =>
total + item.price * item.quantity, 0);
};
```

The `reduce` function iterates through the cart items array to compute a single total value, ensuring the summary is always accurate.



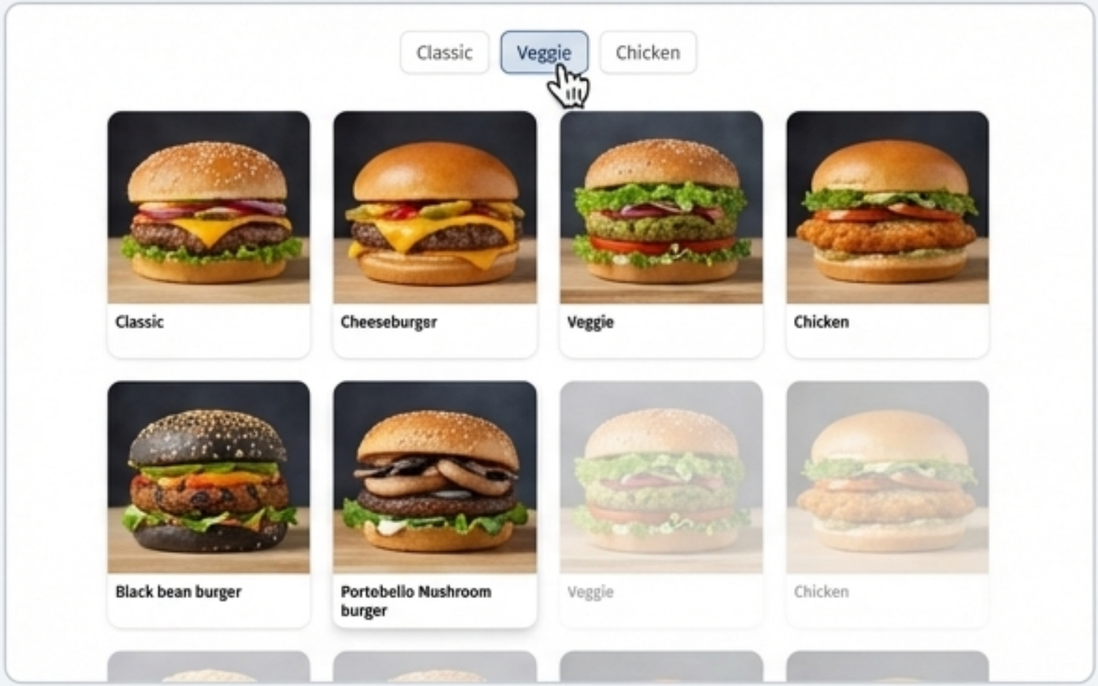
HOFs in Action: Dynamic Rendering with `filter()` and `map()`

`Menu.jsx`

```jsx
// Example snippet from Menu page
const filteredBurgers = MOCK_PRODUCTS.filter(
  (burger) => burger.category === activeFilter
);

return (
  <div>
    {filteredBurgers.map((burger) => (
      <BurgerCard key={burger.id} {...burger} />
    ))}
  </div>
);
```

`filter()` creates a new array based on user selection, and `map()` transforms this array into a list of rendered UI components without direct DOM manipulation.



Proof Point 2: Immutable State Management with ES6

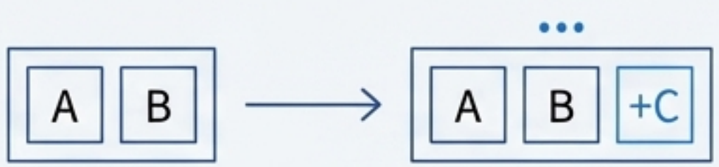
State is managed using complex array structures within React `**useState**` hooks. To ensure predictable state updates and leverage React's performance optimisations, **all state** updates are performed **immutably**.

This principle is critical for:

- Adding or removing items from the shopping cart.
- Updating ingredient lists in the custom burger builder.
- Managing a queue of active notifications.

Key ES6 Feature

The **Spread Operator (...)** is used extensively to create new arrays for state updates rather than mutating existing ones.



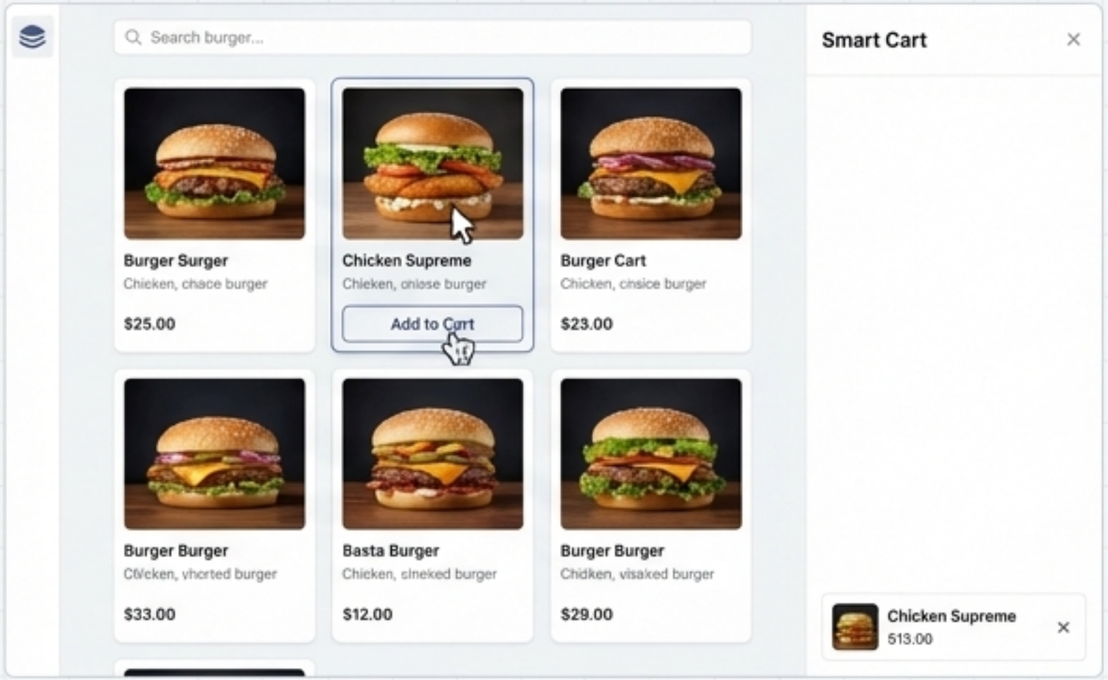
New Array Created: Original is unchanged, ensuring predictability.

State Management in Action: Adding to the Cart

Source Sans Pro
`CartContext.js`

```
// Example snippet for adding an item
const addItemToCart = (newItem) => {
  setCartItems((prevItems) => [...prevItems, newItem]);
};
```

The spread operator `[...prevItems, newItem]` *creates a new array containing all previous items plus the new one. This avoids direct mutation and reliably triggers React's re-rendering process.*

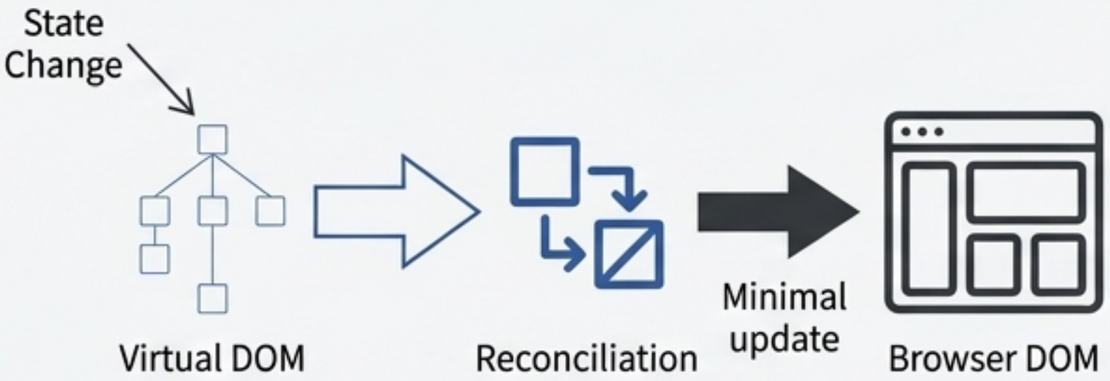


Proof Point 3: High-Performance UI with React's Virtual DOM

Instead of direct DOM manipulation, the application leverages React's reconciliation algorithm. Changes are first made to a lightweight "Virtual DOM," and React then efficiently calculates and applies only the minimal necessary changes to the actual browser DOM.

> **This approach ensures a highly responsive and performant user interface, even with frequent state changes.**

This is demonstrated practically through **Conditional Rendering**, where entire sections of the UI appear or disappear based on application state without reloading the page.



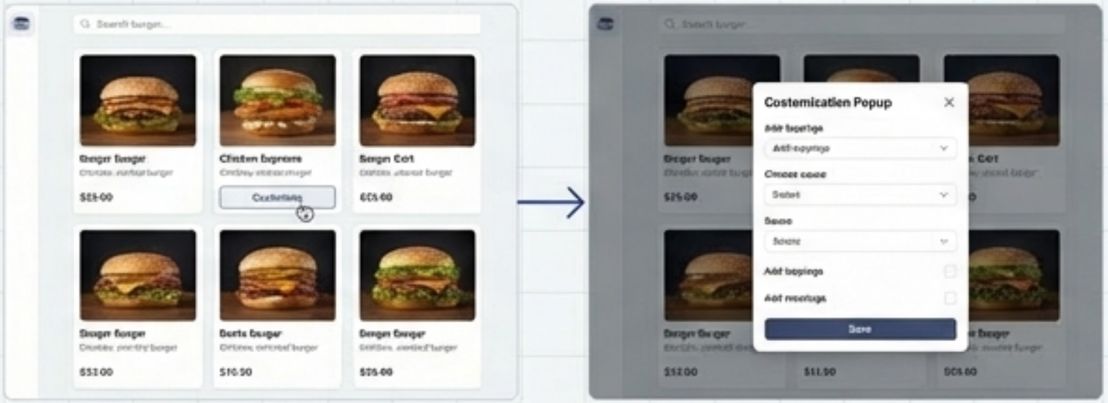
Virtual DOM in Action: Conditional Rendering

`Component.jsx`

```
// Example snippet for a dynamic popup
{isLoading && <LoadingSpinner />}

{isPopupOpen && (
  <CustomizationPopup
    onClose={() => setIsPopupOpen(false)}
  />
)}
```

JSX expressions allow UI elements like loading states or popups to be rendered only when their corresponding state variable is `true`. React handles the efficient adding and removing of these elements from the DOM.

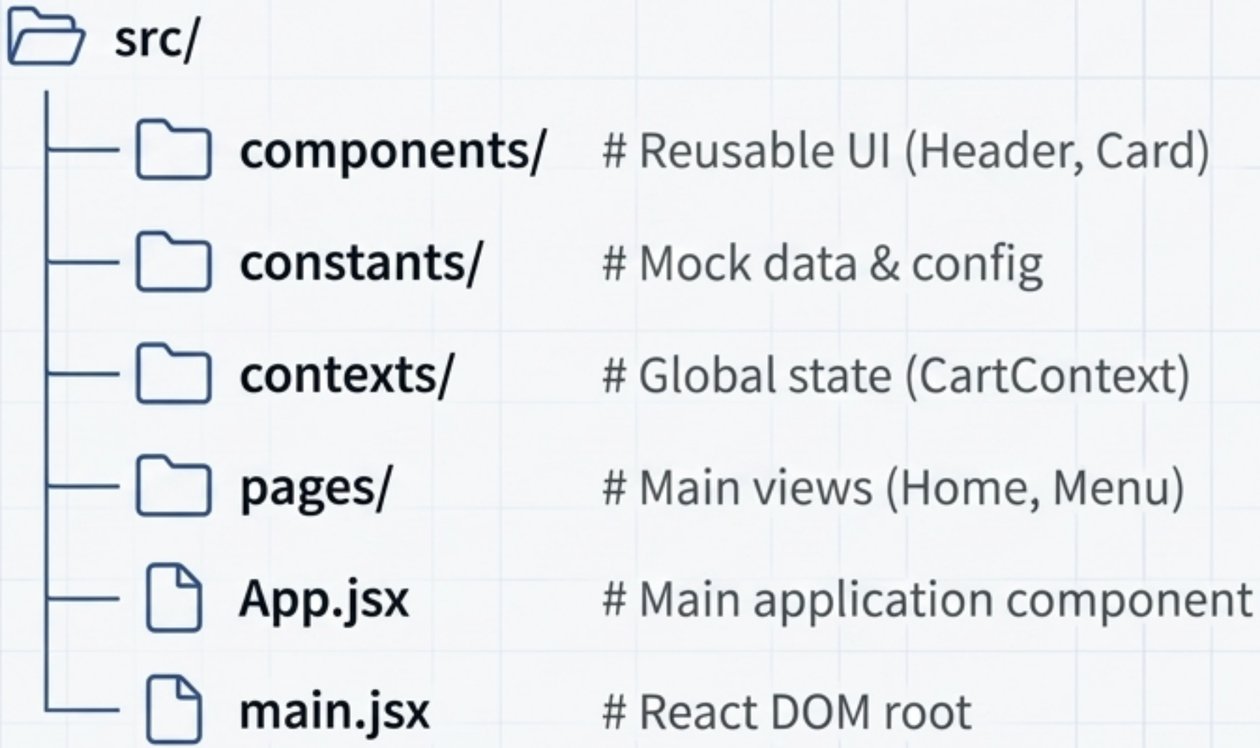


Before

After

The Architect’s Blueprint: A Scalable Project Structure

A well-organised, component-based architecture is essential for maintainability and scalability.
The project structure separates concerns logically.



Componentisation

Reusable components (`Header`, `Card`) promote code reuse.

State Management

Centralised context (`CartContext`) provides global state without prop-drilling.

Separation of Concerns

Clear division between pages, reusable components, and data/logic.

Full Utilisation of Modern JavaScript (ES6+)

The entire codebase is written using modern ES6+ syntax for improved readability, conciseness, and maintainability.

Modules

Component-based architecture with `import` / `export` statements for modular and organised code.

```
import Header from './components/Header';
```

Arrow Functions

Used for concise syntax in components, callbacks, and HOFs.

```
const MyComponent = () ⇒ <div>Hello</div>;
```

Destructuring

Cleanly extracting props and state values, making components easier to read.

```
function Card({ title, description }) { ... }
```

Spread Operator

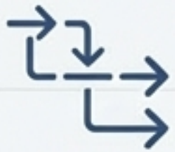
Key for immutable state updates in arrays and objects.

```
setCartItems([...prevItems, newItem]);
```

Objectives Achieved: A Summary of Proficiency

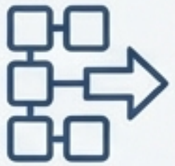
Arrays & Data Structures

Powered the mock database layer (`MOCK_PRODUCTS`) and managed all dynamic state for the cart and notifications using `useState` .



Higher-Order Functions

Implemented `reduce` for cart total calculation, `filter` for the menu system, and `map` for rendering all dynamic UI lists.



ES6+ Compatibility

The entire application is built on a modular architecture using `import/export` , with extensive use of arrow functions, destructuring, and the spread operator.

DOM Manipulation via React

Leveraged the Virtual DOM for high performance and implemented conditional rendering to dynamically show and hide UI elements like popups and loading states.

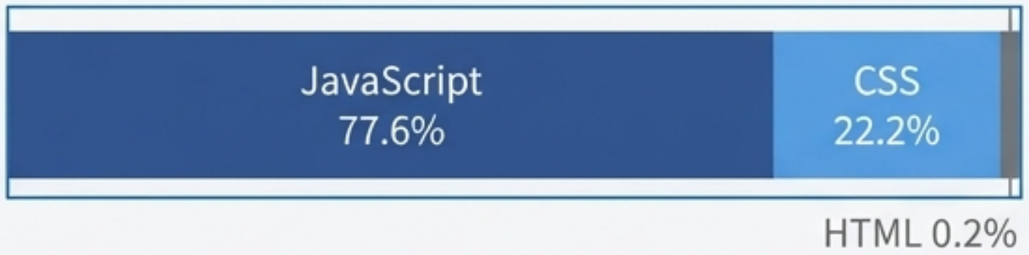
Explore the Implementation

Source Sans Pro
Technology Stack

Framework
React

Build Tool
Vite

Language Distribution



The complete source code is available for review on GitHub. You are invited to run the project locally to experience the application firsthand.

github.com/LABBISRIKANTHBABU/Exposys

Local Setup

```
1. npm install
2. npm run dev
```