🪗 Accordion Menu Documentation (with 🧠 Explanations + 🎨 Styling)

This guide explains the **accordion-style menu** used for displaying **restaurant categories**. The user can expand/collapse different sections to explore menu items in a clean and user-friendly way.

⭐ Core Functionality Overview

✅ What does this accordion menu do?

- 🎯 **Only one section open at a time** – keeps the UI clean
- 🔁 **Toggles** – click again to close an already open section
- 🎞 **Smooth animation** – opening/closing feels polished
- 🔼/🔽 **Arrow icons** – help users understand the current state

---

🔧 Tech Stack Used

- 🔁 `React` (`useState`, `useParams`)
- 🧠 Custom Hook: `useRestaurantMenu`
- 🎨 CSS for animations (`RestaurentMenu.css`)

🧠 State Management: `openSectionIndex`

We use `useState` to track which menu section is **currently open**:

```
const [openSectionIndex, setOpenSectionIndex] = useState(0);
```

- `0`: First section is open by default
- `-1`: No section is open

🔄 Toggle Logic

This function determines which section to open or close:

```
const toggleSection = (index) => {
  setOpenSectionIndex(openSectionIndex === index ? -1 : index);
};
```

- 👆 Click same section? Close it (`-1`)
- 👉 Click a different one? Close current, open new

🧩 Full Accordion Implementation (JSX + Data)

Here's how your `RestaurentMenu.js` maps and renders sections dynamically:

```jsx
{menuSections.map((section, index) => (
  <div className="menu-section" key={section.title}>

    {/* ⭕ Section Title (Clickable) */}
    <div className="menu-section-title" onClick={() => toggleSection(index)}>
      <span>{section.title} ({section.items.length})</span>
      <span>{openSectionIndex === index ? '🔼' : '🔽'}</span>
    </div>

    {/* 📦 Section Content (Shown/Hidden based on state) */}
    <div className={`menu-items-list ${openSectionIndex === index ? '' : 'collapsed'}`}>
      {section.items.map((item) => {
        const info = item.card.info;
        return (
          <div className="menu-item-card" key={info.id}>

            {/* 🖼️ Image */}
            {info.imageId && (
              <img className="menu-item-img" src={CDN_URL + info.imageId} alt={info.name} />
            )}

            {/* 🍽️ Name + Tags */}
            <div className="menu-item-title">
              {info.name}
              {info.isVeg ? <span title="Veg">🥦</span> : <span title="Non-Veg">🍗</span>}
              {info.isBestseller && (
                <span style={{ marginLeft: 8, color: "#ff9800", fontWeight: 700 }}>★ Bestseller</span>
              )}
            </div>

            {/* 📄 Description */}
            <div className="menu-item-desc">
              {info.description
                ? info.description.slice(0, 80) + (info.description.length > 80 ? "..." : "")
                : "No description."}
            </div>

            {/* 💸 Price */}
            <div className="menu-item-price">
              ₹{info.price / 100 || info.defaultPrice / 100 || "-"}
            </div>
          </div>
        );
      })}
    </div>
  </div>
))}
```

🎨 Styling with `RestaurentMenu.css`

Smooth transitions and toggling behavior are handled using these CSS classes:

```css
.menu-section-title {
  cursor: pointer;
  display: flex;
  justify-content: space-between;
  align-items: center;
  font-weight: bold;
  font-size: 1.1rem;
  background-color: #fafafa;
  padding: 12px;
  border-radius: 8px;
  transition: background-color 0.3s ease;
}
.menu-section-title:hover {
  background-color: #f4f4f8;
}

.menu-items-list {
  overflow: hidden;
  max-height: 2000px; /* Large enough to show content */
  transition: max-height 0.5s ease-in-out, padding 0.5s ease-in-out;
  padding: 12px;
}

.menu-items-list.collapsed {
  max-height: 0;
  padding-top: 0;
  padding-bottom: 0;
}
```

🔍 UX Highlights

💧 **Feature**	💡 **Benefit**
🔼/🔽 Icons	Intuitive visual cue for open/closed sections
🎞 Animation	Smoother user experience
❌ Only One Open	Prevents long scrolling & clutter

💧 **Feature**	💡 **Benefit**
♻️ Reusable Code	Easily supports any number of sections

🧪 Pro Tip: Lazy Loading Sections

For better performance on very long menus, consider **loading section content only when it's open** (conditional rendering of the `.map()` loop).

🧾 Summary

☑️ The accordion menu improves UX by organizing large restaurant menus 💡 Simple `useState` + conditional CSS makes it clean & powerful 🔁 Easily scalable and visually intuitive!

🧩 Full Accordion Implementation: JSX + Data 💡 Explained

> This part of the code dynamically renders each menu category and its items. It allows toggling (expand/collapse) of individual sections while managing user interactions and data rendering.

🔁 Looping Through Menu Sections

```
{menuSections.map((section, index) => (
```

- `menuSections`: An array of menu categories like “Starters”, “Main Course”, etc., each containing a list of menu items.
- `map()`: Iterates over all menu sections.
- `index`: Helps track which section is currently active (`openSectionIndex`).

📦 Section Wrapper

```
  <div className="menu-section" key={section.title}>
```

- `key={section.title}`: Unique key to help React track this section efficiently in the virtual DOM.
- `menu-section`: CSS class to style each entire menu block.

⭕ Title Bar (Expandable Header)

```
      <div className="menu-section-title" onClick={() => toggleSection(index)}>
        <span>{section.title} ({section.items.length})</span>
        <span>{openSectionIndex === index ? '🔼' : '🔽'}</span>
      </div>
```

💡 What's going on here?

- `menu-section-title`: Styled with CSS for layout and hover effects.
- `onClick={() => toggleSection(index)}`:
  - Calls the function `toggleSection()`.
  - If the clicked section is already open → it closes.
  - If it's a different one → closes current, opens new.
- First `<span>`: Displays section name and number of items.
- Second `<span>`: Displays an arrow icon to indicate if the section is **open (🔼)** or **closed (🔽)**.

🗂 Section Content Area (Dynamic Collapse)

```
      <div className={`menu-items-list ${openSectionIndex === index ? '' :
'collapsed'}`}>
```

- Dynamically applies the `collapsed` class:
  - If the section **is not open**, `collapsed` class shrinks the section with animation (via CSS).
  - If it **is open**, class is not applied, so it fully expands.

🧾 Mapping Inside Each Section (Items)

```
        {section.items.map((item) => {
          const info = item.card.info;
```

- Iterates over each **menu item** inside the current section.
- Extracts `info` which holds item details like name, price, image, etc.

🖼 Image Display

```
          return (
            <div className="menu-item-card" key={info.id}>
              {info.imageId && (
                <img className="menu-item-img" src={CDN_URL + info.imageId} alt=
  {info.name} />
              )}
```

- `menu-item-card`: Container for each menu item.
- `info.imageId`: If available, shows the image using the Cloudinary CDN.
- Adds a fallback using conditional rendering.

🍽️ Item Title + Tags

```
            <div className="menu-item-title">
              {info.name}
              {info.isVeg ? <span title="Veg">🥦</span> : <span title="Non-Veg">
🍗</span>}
              {info.isBestseller && (
                <span style={{ marginLeft: 8, color: "#ff9800", fontWeight: 700
}}>★ Bestseller</span>
              )}
            </div>
```

- Displays:
  - Item name (`info.name`)
  - 🥦 for vegetarian, 🍗 for non-veg based on `info.isVeg`
  - "★ Bestseller" tag if `info.isBestseller` is true

🎯 This gives instant visual cues for food type and popularity.

📄 Description Preview

```
            <div className="menu-item-desc">
              {info.description
                ? info.description.slice(0, 80) + (info.description.length > 80 ?
"..." : "")
                : "No description."}
            </div>
```

- Shows the first 80 characters of description, followed by `...` if it's longer.
- Provides clean, short summaries.
- Fallback: If no description, shows "No description."

💸 Price Display

```
            <div className="menu-item-price">
              ₹{info.price / 100 || info.defaultPrice / 100 || "-"}
            </div>
```

- Most Swiggy prices come in *paise* → so divide by 100.
- If `info.price` isn't available, fallback to `info.defaultPrice`.
- Fallback again: Show "-" if neither exist.

📦 Result: Interactive Accordion with Data-Driven Menu

✅ Every part of the UI is **driven by real data** from the backend ✅ Responsive to **user interaction** using local state ✅ Cleanly separated into **title bar** (clickable) and **content** (dynamic)

🎯 Visual Summary of Behavior

Part	Behavior
`menu-section-title`	Click to toggle open/close state of section
Arrow icons	🔼 = open, 🔽 = closed
Section content	Shown/hidden using CSS + `openSectionIndex` comparison
`map()` on items	Dynamically renders each menu item with image, tags, price
CSS animations	Smooth expand/collapse transitions via `max-height` property

---

🎯 Goal: Accordion with Two Variants

✅ **1. Controlled by Parent** — All logic (open/close) is managed by the **parent component**.

✅ **2. Controlled by Each Child** — Each accordion item manages its **own open/close state** internally.

🧩 1️⃣ Accordion Controlled by **Parent**

📌 Use-case: Ideal when **only one section** should be open at a time (like a restaurant menu).

✅ Features:

- Parent manages `openIndex`.
- Only one accordion item is expanded at a time.
- Clean separation of logic.

🧪 Code

```jsx
// AccordionParentControlled.jsx
import React, { useState } from "react";

// ➕ Accordion Item as Child
const AccordionItem = ({ title, content, isOpen, onToggle }) => {
  return (
    <div className="border mb-2 rounded shadow">
      <div
        className="bg-purple-100 px-4 py-2 cursor-pointer flex justify-between items-center"
        onClick={onToggle}
      >
        <span>{title}</span>
        <span>{isOpen ? "🔼" : "🔽"}</span>
      </div>
      {isOpen && (
        <div className="bg-white px-4 py-2 transition-all">
          {content}
        </div>
      )}
    </div>
  );
};

// ⭕ Parent Controls Open State
const AccordionParentControlled = () => {
  const [openIndex, setOpenIndex] = useState(null);

  const data = [
    { title: "React", content: "React is a JavaScript library for building UI." },
    { title: "Vue", content: "Vue is a progressive JavaScript framework." },
    { title: "Angular", content: "Angular is a platform for building mobile and desktop web apps." }
  ];

  const handleToggle = (index) => {
    setOpenIndex(openIndex === index ? null : index); // toggle or close if already open
  };

  return (
    <div className="max-w-md mx-auto mt-6">
      <h2 className="text-xl font-bold mb-4">🚀 Accordion (Parent Controlled)</h2>
      {data.map((item, index) => (
        <AccordionItem
          key={index}
          title={item.title}
          content={item.content}
          isOpen={openIndex === index}
          onToggle={() => handleToggle(index)}
        />
      ))}
    </div>
  );
};

export default AccordionParentControlled;
```

🧾 Explanation

Line	Description
`useState(openIndex)`	Maintains the currently open accordion item
`AccordionItem`	Pure component, doesn't manage its own state
`onToggle()`	Instructs parent to change `openIndex`
`openIndex === index`	Only one open item based on index match
🔼/🔽	Visual cue for open/closed state

🧩 2️⃣ Accordion Controlled by **Each Child**

📌 Use-case: Ideal when **multiple items** can be opened at once.

🧪 Code

```
// AccordionChildControlled.jsx
import React, { useState } from "react";

// ⚙️ Each child manages its own state
const AccordionChildItem = ({ title, content }) => {
  const [isOpen, setIsOpen] = useState(false);

  const toggle = () => setIsOpen((prev) => !prev);

  return (
    <div className="border mb-2 rounded shadow">
      <div
        className="bg-green-100 px-4 py-2 cursor-pointer flex justify-between
items-center"
        onClick={toggle}
      >
        <span>{title}</span>
        <span>{isOpen ? "🔼" : "🔽"}</span>
      </div>
      {isOpen && (
        <div className="bg-white px-4 py-2 transition-all">
          {content}
        </div>
      )}
    </div>
  );
};

const AccordionChildControlled = () => {
  const data = [
    { title: "HTML", content: "HTML defines the structure of your web content." },
    { title: "CSS", content: "CSS styles your HTML content." },
    { title: "JavaScript", content: "JavaScript makes your page interactive." }
  ];

  return (
    <div className="max-w-md mx-auto mt-6">
      <h2 className="text-xl font-bold mb-4">📦 Accordion (Child Controlled)</h2>
      {data.map((item, index) => (
        <AccordionChildItem key={index} title={item.title} content={item.content} />
      ))}
    </div>
  );
};

export default AccordionChildControlled;
```

🧾 Explanation

Line	Description
`useState(isOpen)`	Each child has its own `isOpen` state
`toggle()`	Each child toggles its own state independently
Multiple open allowed	Because no shared state is used
Reusable	More decoupled, easier for large dynamic components

⚖️ Comparison Table

Feature	Parent Controlled	Child Controlled
Who manages open state?	Parent	Each child
Only one open at a time	✅ Yes	❌ No (multiple can be open)
Reusable logic	Centralized	Decentralized
Best for...	Step-by-step or exclusive menus	FAQs, checklists, multi-open cases

---

🧠 Use-Cases Summary

- **Use Parent-Controlled** when:
  - You want **only one section** open at a time (e.g., restaurant menus, form steps).
- **Use Child-Controlled** when:
  - You want **multiple items open** at once (e.g., FAQ pages, filter dropdowns).

🚀 Reusable Accordion Component (with `singleOpen` mode toggle)

```
// SmartAccordion.jsx
import React, { useState } from "react";

// 🧱 Single Accordion Item
const AccordionItem = ({ title, content, isOpen, onToggle }) => {
  return (
    <div className="border rounded mb-2 shadow">
      {/* 🔘 Clickable Title */}
      <div
        className="bg-indigo-100 px-4 py-2 cursor-pointer flex justify-between items-center"
        onClick={onToggle}
      >
        <span>{title}</span>
        <span>{isOpen ? "🔼" : "🔽"}</span>
      </div>

      {/* 📂 Conditional Content */}
      {isOpen && (
        <div className="bg-white px-4 py-2 transition-all">
          {content}
        </div>
      )}
    </div>
  );
};

// 🔁 Main Reusable Accordion Component
const SmartAccordion = ({ data = [], singleOpen = false }) => {
  // 📦 State
  const [openIndex, setOpenIndex] = useState(null); // for singleOpen
  const [openStates, setOpenStates] = useState(() => data.map(() => false)); // for multi-open

  // 🔁 Toggle Logic
  const handleToggle = (index) => {
    if (singleOpen) {
      setOpenIndex(openIndex === index ? null : index);
    } else {
      setOpenStates((prev) =>
        prev.map((isOpen, i) => (i === index ? !isOpen : isOpen))
      );
    }
  };

  return (
    <div className="max-w-md mx-auto mt-6">
      <h2 className="text-2xl font-bold mb-4 text-center">
        {singleOpen ? "⭕ Accordion (Single Open Mode)" : "📦 Accordion (Multi
Open Mode)"}
      </h2>

      {data.map((item, index) => (
        <AccordionItem
          key={index}
          title={item.title}
          content={item.content}
          isOpen={singleOpen ? openIndex === index : openStates[index]}
          onToggle={() => handleToggle(index)}
        />
      ))}
    </div>
  );
};

export default SmartAccordion;
```

📄 Usage Example

```
// App.jsx
import React from "react";
import SmartAccordion from "./SmartAccordion";

const faqData = [
  {
    title: "🍕 What is React?",
    content: "React is a JavaScript library for building user interfaces."
  },
  {
    title: "⚛️ What is a Hook?",
    content: "Hooks let you use state and other features without writing a class."
  },
  {
    title: "🔁 What is useEffect?",
    content: "It's used for side effects in functional components."
  }
];

const App = () => {
  return (
    <>
      {/* 👇 Parent-controlled (Only one open) */}
      <SmartAccordion data={faqData} singleOpen={true} />

      <hr className="my-10" />

      {/* 👇 Child-controlled (Multiple can open) */}
      <SmartAccordion data={faqData} singleOpen={false} />
    </>
  );
};

export default App;
```

🎨 Optional Tailwind Styling Guide

```
/* optional styles if not using Tailwind */
body {
  font-family: sans-serif;
  background: #fdfdfd;
}
```

📊 Feature Summary

Feature	☑ Supported
Parent-controlled mode	☑ Yes
Child-controlled mode	☑ Yes
One component handles both	☑ Yes
Dynamic content	☑ Yes
Icons + Clean layout	☑ Yes
Reusable + Easy to Extend	☑ Yes
