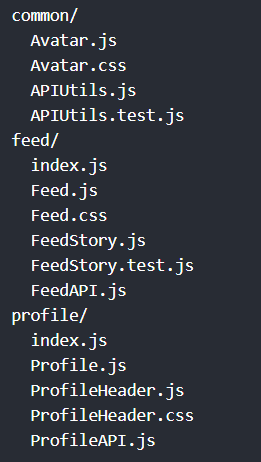
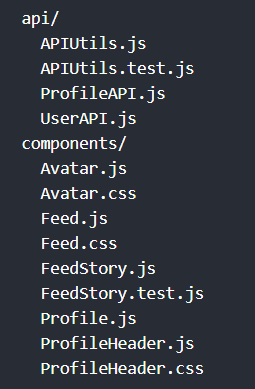
Chapter 5 – Let’s get hooked

**Before starting with hooks, let’s talk about React folder structure.**

Is there a recommended way to organize React projects?

React doesn’t impose any specific way to organize your files into folders.

Grouping files by **FEATURES** Grouping files by file **TYPES**

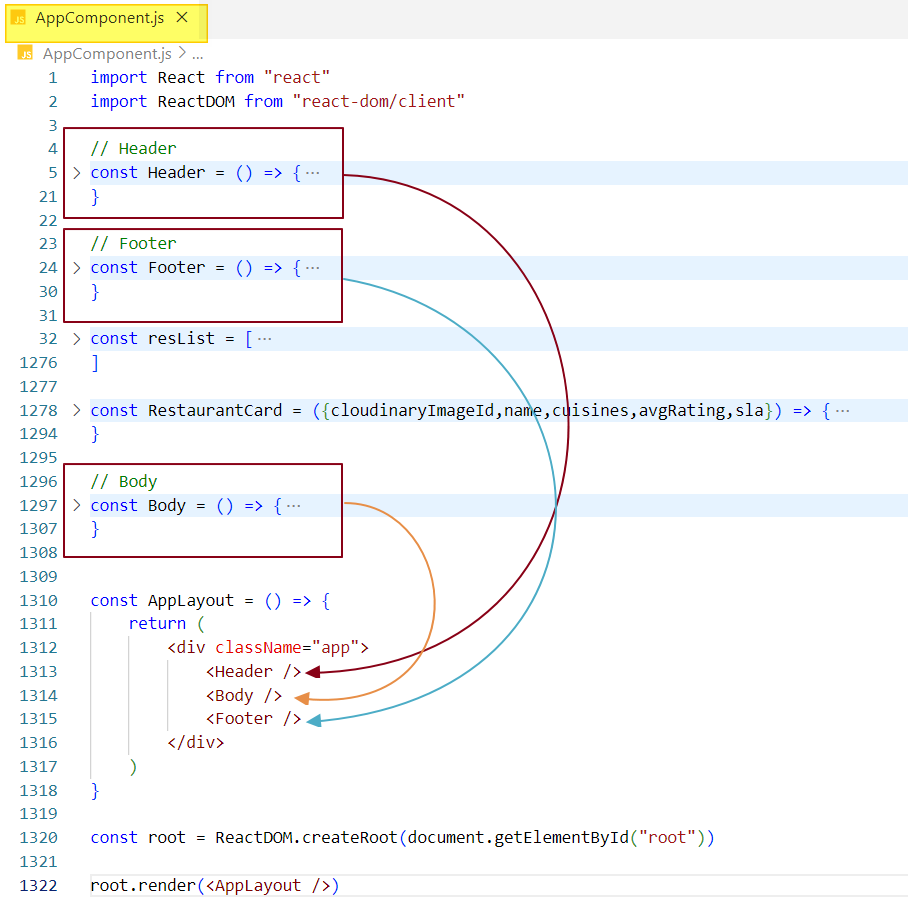


We can create our own custom folder structure to keep the project organized. But before doing that, let’s first understand an important design pattern in React: The **Single Responsibility Principle (SRP).**

Single Responsibility Principle (SRP)

The Single Responsibility Principle (SRP) is a design pattern that states each component should have only one specific responsibility or purpose.

In our application, we created three components - header, footer, and body - inside the **AppComponent**. This violates SRP because the **AppComponent** is doing more than one task: first, it creates the header, footer, and body components, and second, it renders them inside the **AppLayout** component.



To follow the SRP, each component should be responsible only for its specific role. Therefore, we’ll create separate files for the header, footer, and body components. Later, we can import these components into the **AppLayout** component as needed.

Let’s restructure our project according to common industry practices, where all the code is organized within a **‘src’** folder. While it's not mandatory to use a **‘src’** folder, we’ll follow this approach to align with industry standards.

It’s best to create separate files for each component. We have the following components:

1. **Header**
2. **RestaurantCard**
3. **Body**
4. **Footer**

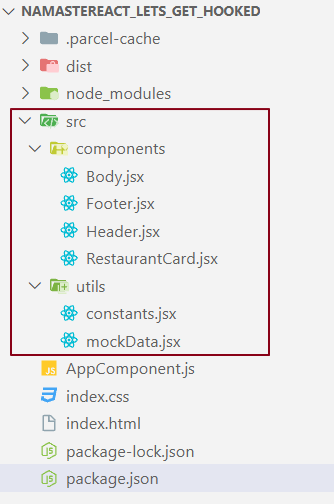
We’ll place these components inside a **‘components’** folder, which will be a subfolder within the **‘src’** folder. When naming component files, always start with a capital letter. In this project, we are using the **.js** extension, but **.jsx** can also be used based on your preference.

We will also create a **config** or **constants** file in our application to store hardcoded information. This file is typically placed in a **‘utils’** folder, as per industry standards. The reason for this is that these files are considered utilities, providing useful configurations or resources for the application.

Additionally, we will create a **mockData** file inside the **‘utils’** folder to store hardcoded data for the restaurant list.

**Note:** In some cases, developers use **‘common’** instead of **‘utils’** for the folder name.

Current folder structure



Import and Export in React

**Import**:  
The import statement in a file allows us to use the contents of another file that has been exported. There are two types of imports in React:

1. Named Import: Imports specific exports from a module.
2. Default Import: Imports the default export from a module.

**Export**:  
The export statement in a file allows us to make the contents of that file available to other files. There are two types of exports in React:

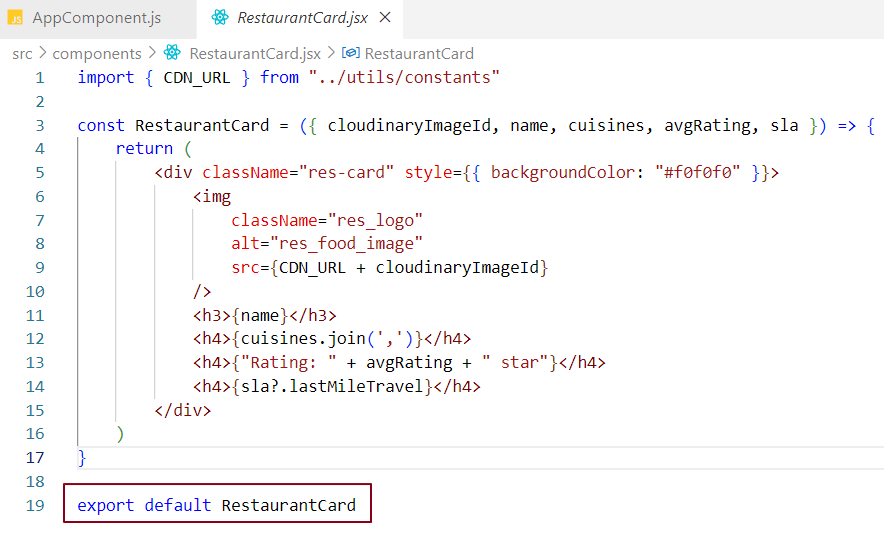
1. Named Export: Exports specific variables or functions that can be imported by name.
2. Default Export: Exports a single value (variable, function, or class) from a module as the default export.

Let’s consider a few examples of exports and imports.

Default Export - (Export Single component)

A **default export** in React is a way to export a single value from a module, which can be a component, function, or variable, allowing it to be imported without curly braces.

syntax - export default <componentName>;

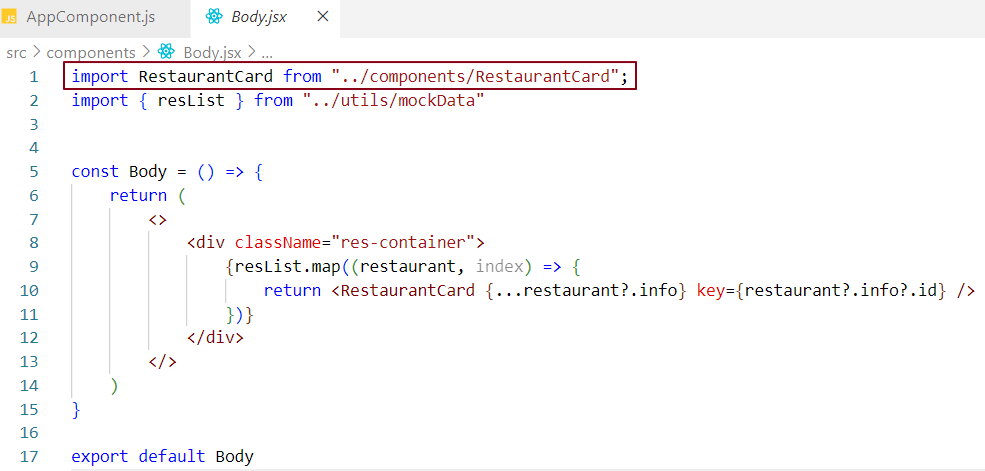


Default Import - (Import single component)

A **default import** in React is used to import a single exported value from a module, allowing you to use any name for that value in the importing file.

Syntax - import <componentName> from <FilePath>;

In the below code snippet, the **RestaurantCard** component is being imported into the **Body.js** file. While importing, we can include the **.js** extension in the file path without affecting the functionality.



We can use any name for the component in a default import. For example, if **RestaurantCard** is changed to **XYZ**, it won’t cause any errors, but make sure to use **XYZ** everywhere in the **Body.js** module.



Named Export – (Export Single component)

A named export allows us to export single or multiple values from a module by their names, enabling them to be imported using those names.

Named export syntax –

To export specific values from a module, use the following syntax:

// Example: Exporting multiple values

export const value1 = 'Hello';

export const value2 = 'World';

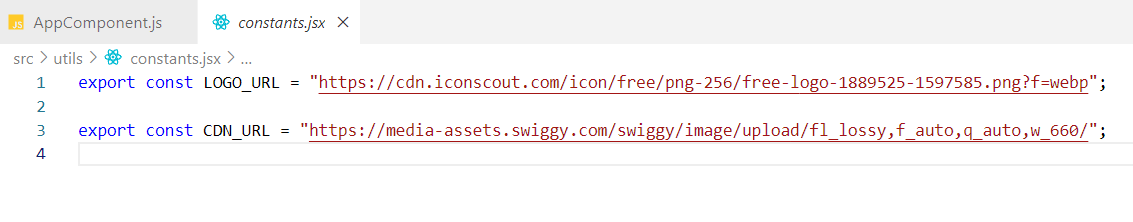
we can also export them at the end of the file:

const value1 = 'Hello';

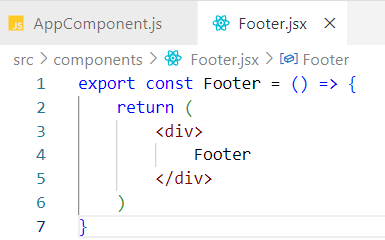
const value2 = 'World';

export {value1, value2};

Exporting constant values using Named export syntax -



Exporting a component using Named export syntax -



Named Import – (Import Single component)

A named import allows you to import specific values from a module by their exact names, using curly braces.

Named import syntax - import {value1, value2…...} from <FilePath>;



**Note:** {componentName} in this context is not object destructuring.

Default Export - (Export multiple components)

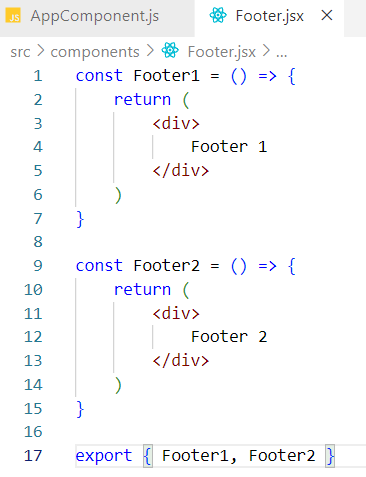
Not possible

Default Import - (Import multiple components)

Not possible

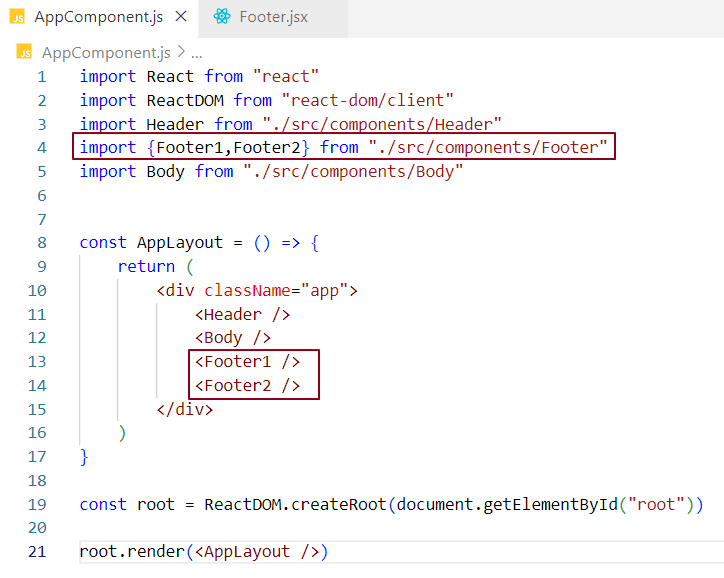
Named Export - (Export Multiple components)

Exporting two-footer components in react.



Named Import – (Import Multiple components)

Importing two footer components in react.



Another way -



We can mix default and named imports and exports, but the same rules apply.

In the previous chapter, we built a skeleton of our application. Now, let's add functionality to it. We'll create a button labelled "Top-Rated Restaurants." When this button is clicked, it should filter out the top-rated restaurants and display them in the UI.

Inside the Body component, let's add a button labelled "Top Rated Restaurants" above the restaurant list. We'll attach an onClick event handler to the button, which will filter and display restaurants with an average rating greater than 4.0 when clicked.



In the code snippet, when we click the "Top Rated Restaurants" button, the onClick event handler is invoked, which filters the list of restaurants. Once the filtering is complete, it updates the listOfRestaurants variable with the filtered data and logs this data to the console. At this point, we expect the filtered data to appear in the UI because we are using the map function on this filtered listOfRestaurants. However, there are no visible changes in the UI because the data layer is not synced with the UI layer since we are using a regular JavaScript variable listOfRestaurants. To sync the UI layer with the data layer, we need powerful variables such as React state variables.

Why do I need state or state variable when I have local JS variable?

React only tracks state variables; it does not monitor local variables. In our code, if the local variable listOfRestaurants is modified by a function, react will not be aware of this change. Although the variable may be used in multiple places, react won’t recognize why it was modified and won’t update the UI to reflect these changes because local variables are stateless. They do not maintain their own state, and as we know, react only tracks state variables.

**Note:** To ensure that our variables remain in sync with the UI, we need to use state variables. This allows React to monitor these state variables and maintain their state. We can create state variables using the useState hook.

**What is State in React?**

State in React is like a special storage space for a component. It helps keep track of information that can change over time.

1. **Storage for Data:** State is an object that holds data relevant to a component. For example, it can store user input, the status of a button, or the number of items in a cart.
2. **Dynamic Updates:** The data in state can change. When something happens, like a user clicking a button, the state can be updated to reflect that change.
3. **Re-renders the UI:** When the state changes, react automatically updates the component’s UI to show the new information. This means that you don’t have to manually change what the user sees; React does it for you.
4. **Local to Components:** Each component can have its own state. This means different components can have their own separate data, which helps keep things organized.

Example: Imagine a simple counter app:

* **State:** Holds the current count (e.g., 0).
* **Button Click:** When you click a button to increase the count, the state updates to the new number (e.g., from 0 to 1).
* **UI Update:** The display automatically shows the new count without you having to refresh or change anything manually.

What is a Hook?

A standard JavaScript function created by Facebook developers that provides various utility functionalities is known as a hook. There are different types of hooks, and one of the most common ones is the useState hook.

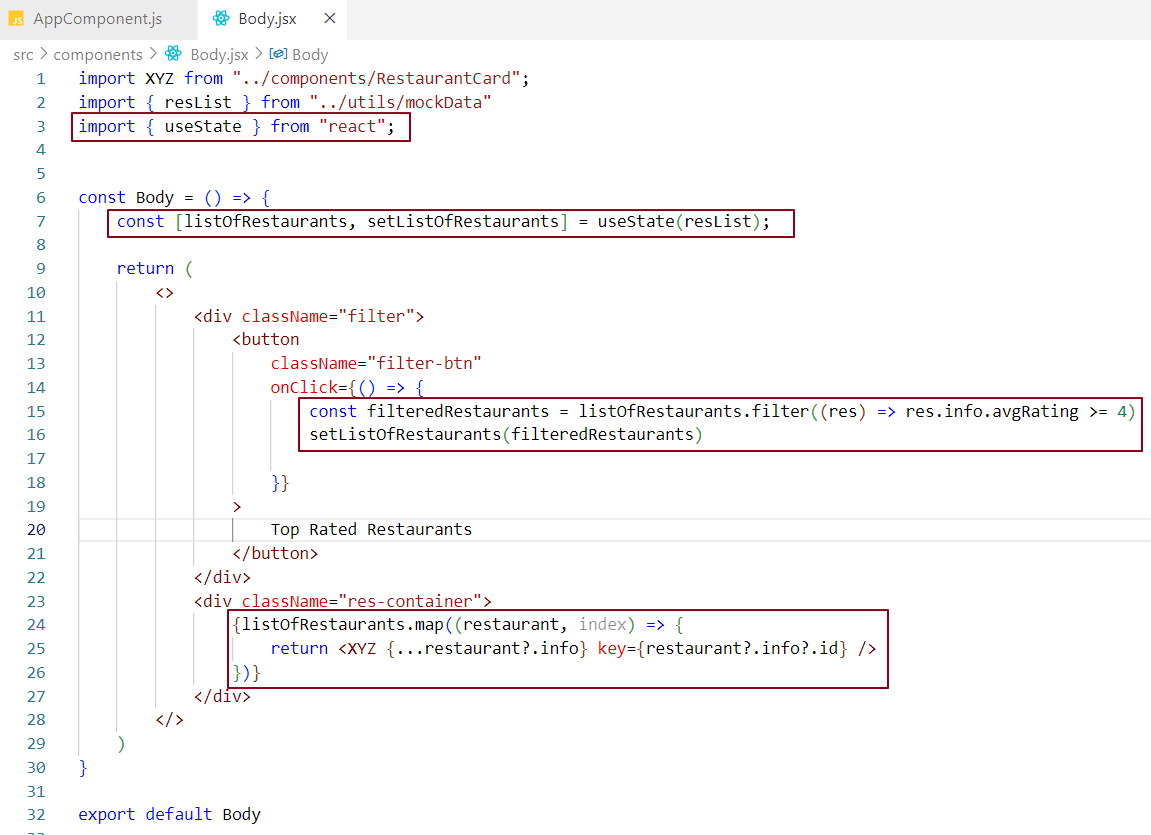
What is a useState Hook?

The useState hook is a function provided by React that allows you to create state variables in a functional component.

The useState hook returns an array where the first element is the name of the state variable, and the second element is a setter function used to update the variable.

The useState function expects an initial value as an argument to set up the state variable.

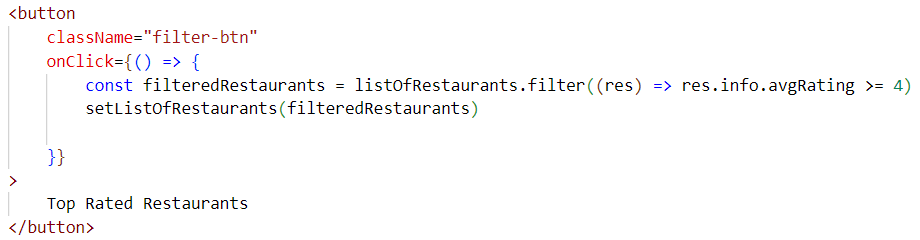
Using react state variable using useState hook





listOfRestaurants - This is the state variable that holds the list of restaurants, starting with the value of resList.

setListOfRestaurants - This function updates the listOfRestaurants. When you call it with a new value, the component will re-render to show the updated list.

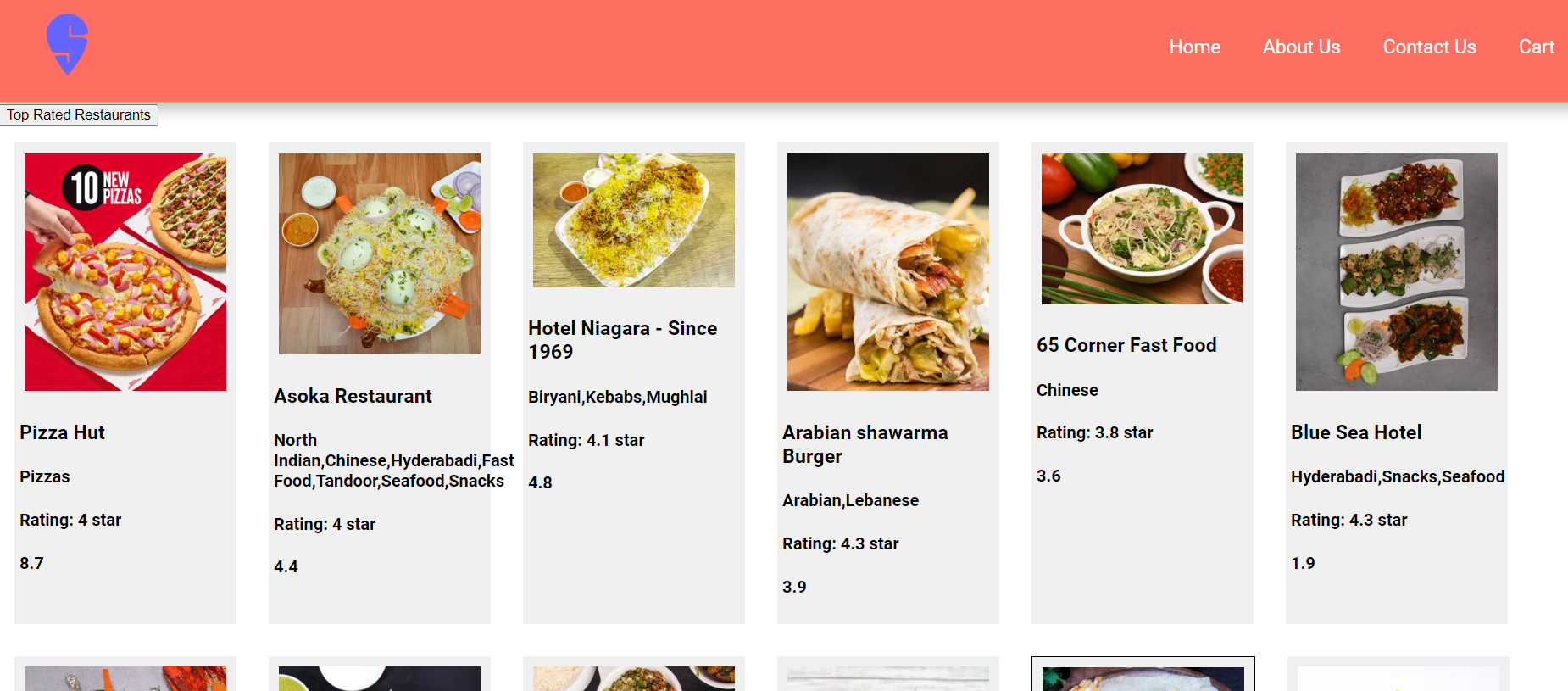


Clicking the button Top Rated Restaurants triggers the onClick event handler, which performs the following actions.

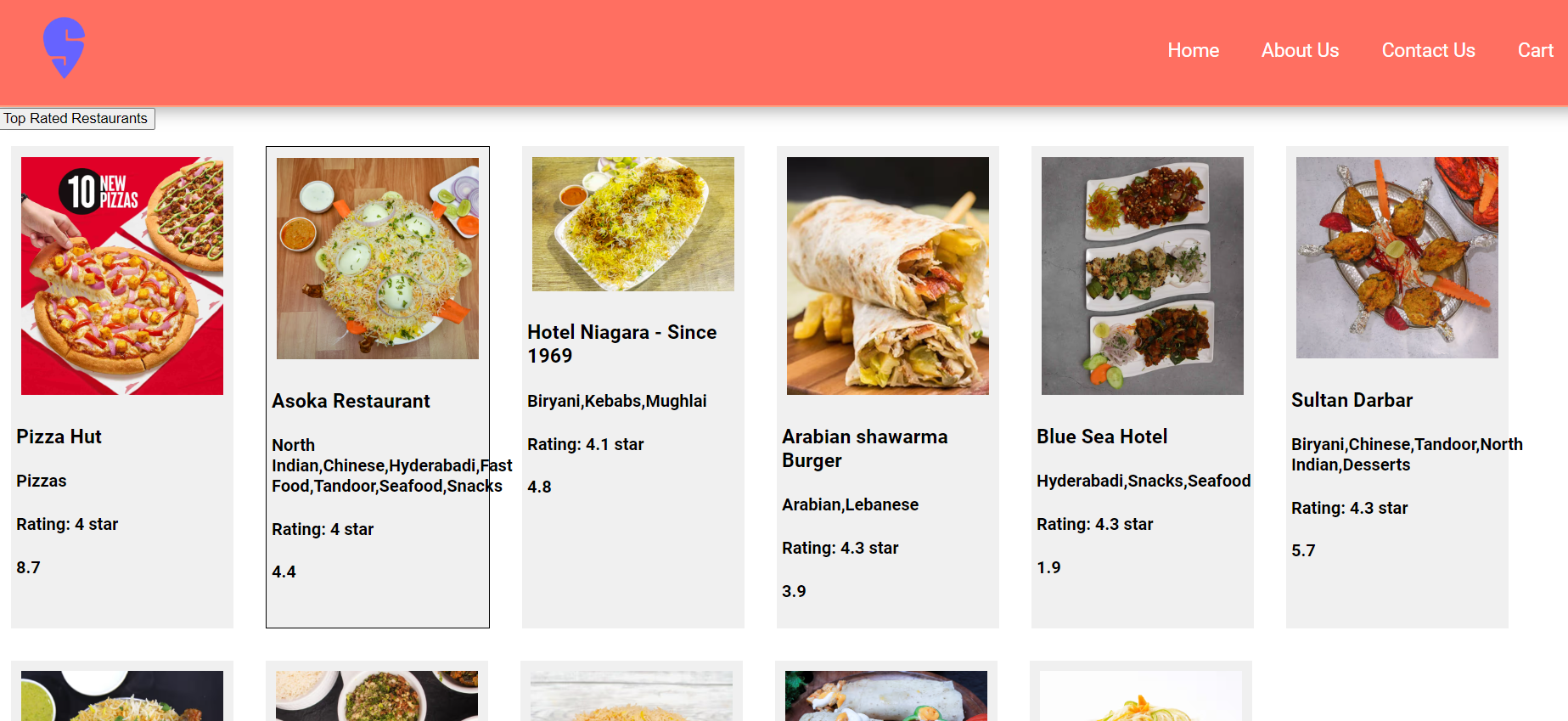
1. Filtering: The code filters listOfRestaurants to retain only those restaurants with an average rating of 4 or higher.
2. Updating State: The setListOfRestaurants function updates the state with the filtered list.
3. Reconciliation: React re-renders the component, checking for changes in the state and updating the UI to display the new list of top-rated restaurants.

**Note 1 –**

When the useState setter function is invoked or a component's state changes, react triggers a reconciliation process to identify differences between the current and previous virtual DOMs. It updates only the changed portions of the UI during the render cycle. Whenever state variables update, react re-renders the component, promptly syncing the UI with the data layer. To achieve this efficiency, react uses a reconciliation algorithm, known as the diffing algorithm or React-Fibre, which efficiently detects UI changes.



After clicking on Top Rated Restaurants button



**Note 2 -**

In React, when you create elements, you're actually creating virtual DOM objects. These virtual copies are kept in sync with the real DOM through a process called "Reconciliation" or the "diffing" algorithm. During each rendering cycle, react compares the new virtual DOM (the updated version) with the old virtual DOM (the previous version) and makes only the necessary changes to the actual DOM. This efficient process ensures that updates happen quickly, improving performance.