Common Questions

Q. **Why do we use React? Some of us might wonder why we don't just stick to HTML, CSS, and JAVASCRIPT for everything we've been doing?**

A. Of course! It's absolutely possible to accomplish everything using regular HTML, CSS and JAVASCRIPT without using REACT.

- However, we choose React because
  - It enhances our developer experience, making it more seamless and efficient.
  - It makes you write less code and do more on the web page, also optimizes somethings on web page. So that, Things happen very fast, this is what the major job of a UI Library or framework.

> It's best to create separate files for each component.

1. How can we achieve this?
   - To achieve this, Let's discuss the folder structure.
   - We are going to Restructure our project folder "NAMASTE-REACT"
   - There is a very good convention in the industry that all the code in a React project is kept in a `src-folder`, there is no compulsion to use a `src-folder` in a project. But here we are following what the industry follows.
   - We are creating and moving our App.js file in the `src-folder` and whatever new files we create we put them in the `src-folder`.

> **NOTE: Don't forget to update the path of the `App.js` file in `index.html` otherwise we will get an Error.**

- The best practice is to make separate files for every component.

  We have the following components.

```
1. Header
2. RestaurantCard
3. Body
```

- We put all the above components inside the folder named `components`(child-folder) which has been placed inside the `src- folder`(parent folder). When we are creating separate component files inside the `components-folder` always start with a capital letter like.

In this course, we are using (.js) as an extension.

```
1. Header.js
2. RestaurantCard.js
3. Body.js
```

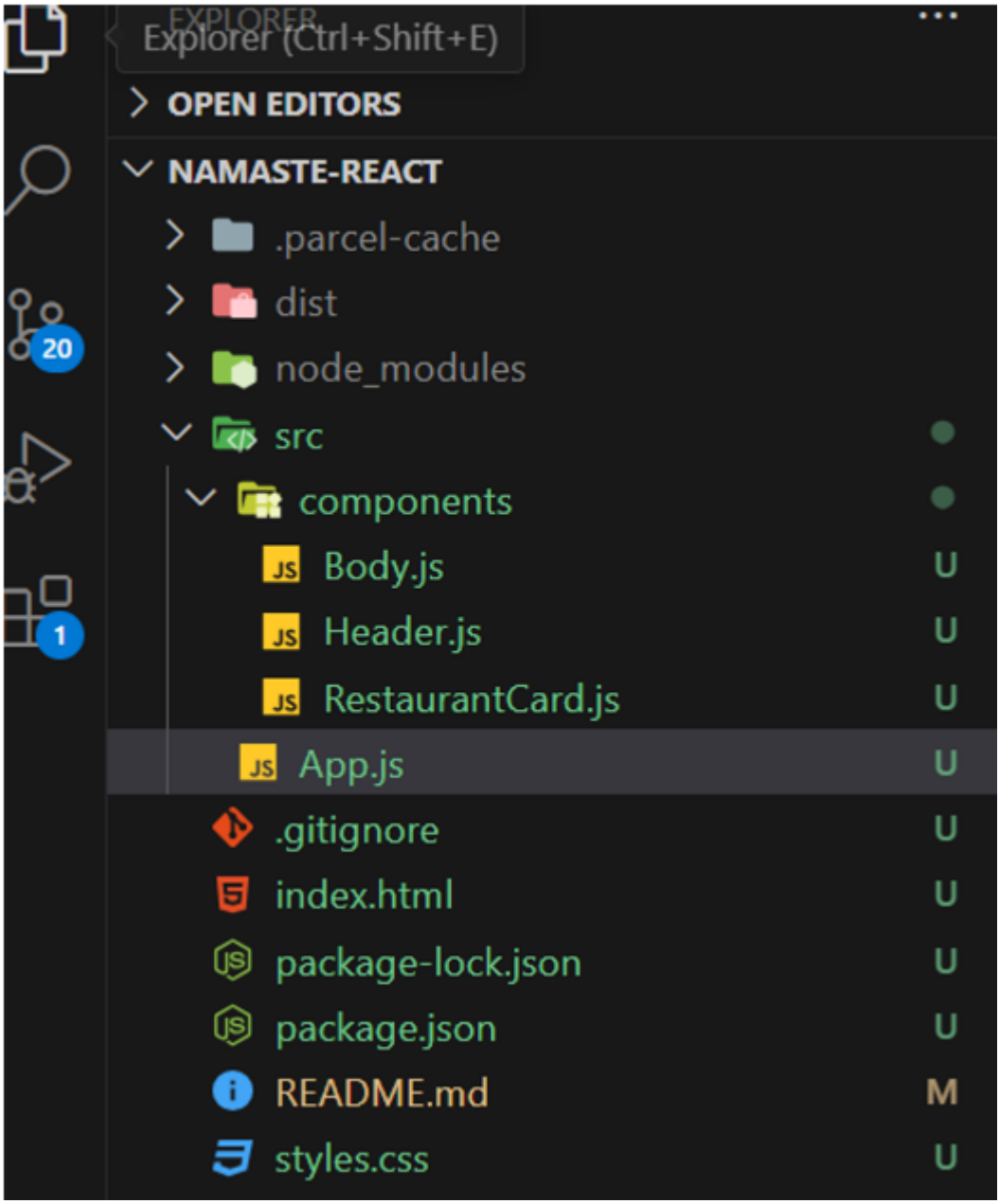
> NOTE: We can use (.jsx) as an extension instead of (.js) it's up to the developer's wish.

```
1. Header.jsx
2. RestaurantCard.jsx
3. Body.jsx
```

> `.tsx` stands for typescript.

> 📢 NOTE: NO Thumb Rule to use this convention, we could use any folder structure you wish.

Example of folder structure :



Understanding Export and Import in React

Two types of export/import in React, We will understand each of them in details.

1. Default export/import.

2. Named export/import.

As we know we move each component's code and create new files individually. Still, Our React project throws an error because our `App.js` file doesn't have components in it and we are using components inside the `App.js` to solve this we need to import the components from their respective files which have been kept inside the `components-folder` in src.

to understand it well let's take an example of the `Header.js` component.

Example:

The **Header.js** component is missing in **App.js** so we are not able to use it, if we try to use it, It throws an error, To solve this, here we have to *import* **Header.js** inside the **App.js** and before the *import*, we have to 1st initially *export* **Header.js** component.

1. **Default export/import.**

Step-1 (export)——————————>

We use the `'export'` and `'default'` keywords with the component name at the end of the component file. Understand the Syntax properly.

Export example :

```
export default Header;
```

or

```
export default Header.js;
```

Step-2 (import)——————————>

Inside `App.js` We use `'import'` with the component name at the start of the file. Understand the Syntax properly.

Import Example :

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

2. **Named export/import.** We use it , when we have to import/export multiple files.

Named Export Example : In the name export, we just `export` keyword before `const`.

```
export const RES_IMG_CDN =
  "https://media-
assets.swiggy.com/swiggy/image/upload/fl_lossy,f_auto,q_auto,h_600/";
```

Named Import Example : In the named import, we use `curly bracket`.

```
import { RES_IMG_CDN } from "../utils/constants";
```

But there's a catch: If we intend to export multiple items simultaneously, from single file `'default export/import'` won't work; it'll result in an error. For instance, in our `' constants.js '` file, we've stored URLs for logos and images using separate variables, which means default export/import won't be feasible. Instead, we can employ `'named export/import'` to handle this scenario effectively.

3. **Can we use default export with named export ?**
    - yes

> 📢 NOTE:

```
1. If you are using Vs-Code Editor during
import it automatically tracks the path of the component
and gives suggestions to us. so we don't have to worry
about the path.
2. We don't have to put an extension in the file in the
import statement. If want to put then completely Fine.
```

> Important Note : `resList` contains hard-coded data and we never put any hard-coded data like the source-URL of logo and images inside the component file. That's not the best practice the industry follows. **Never ever keep Hard-coded data in the component files.**

1. where should we keep the hard-coded files ?

    - Created a new folder within the `'src'` folder called `'utils'`

        > (Utils means common utilities that can we use across all our app & some people named it as "common", "config" etc. but follow industry standard.)

        , and inside it, we've created a file named `'constants.js'`

        ```
        constants.js
        ```

        ```
          why this file name is in samll letter?
          - because, this is not a component,
          - differentiate between component and utilities we use naming
        conventions.
        ```

to store all hard coded data. You can choose any name for this file, but we've opted for lowercase letters since it's not a component. Additionally, we've included the source of images in this file.

- We've stored our mock data for `' resList '` in a file named `' mockData.js '`, which resides within the `'utils'` folder.
- We need to export data from `' constants.js '` and `' mockData.js '`, and then import it into the necessary component files where it will be used

> 2. It's good practice to keep our component clean and small, make sure our component file should not exceed more than 100 no of lines code.

Event Handler

Let's make our app more lively and engaging.

In this tutorial series, we're learning by actually building things. As we go along, we'll keep adding new features to our app.

For now, we're going to add a button. When you click on this button, it will show you the best-rated restaurants.

Inside the body component, We'll add button and give it any class name we want.

We are adding `onclick event handler` inside the button, `onclick` has `call-back function` which will be called when we clicked on the button, there is condition when we clicked on the restaurant we will get a restaurant which avg-rating is more than 4.2 (avg-rating > 4.2)

1. How these cards are coming on the screen?
    - These cards are appear on the screen because we're using the `map()` method to go through each restaurant in our mock data (`resList`). This method helps us display the information from each restaurant on individual cards. so any thing changes in reslist the cards displayed on the screen will also change.
    - We're simplifying our mock data to improve our understanding, or you can use the data provided below (which is the actual data from the Swiggy API)
    - We utilize the data of `resList`, incorporating a condition within the callback function. This condition triggers when we click on the button, displaying only the restaurants whose average rating is above 4.2 . That's what we are expecting.
    - Despite implementing the below code,

```
<button
    type="button"
    class="btn btn-primary"

    onClick={() => {
        const filteredList = listOfRestaurants.filter(
          (res) => res.info.avgRating > 4.2
```

```
          );
          console.log(filteredList)
        }}
>
    Top Rated Restaurant
</button>
```

no UI changed on the screen. However, we successfully filter the data, which we can confirm by checking the filtered results using `console.log(listOfRestaurants)`.

> 📢 Note: whenever we have react App, we have a UI layer and data layer, UI layer will display what is being sent by the data layer.

2. How can we display filtered restaurants dynamically on UI(display screen) ?
    - Here, we're utilizing data retrieved from the `listOfRestaurant` variable, which stores an array of objects. It's treated as a regular variable within our codebase.
    - However, for this functionality, we require a superpowerful React variable known as a `'state variable'`.
3. Why is it called state variable?
    - It maintains the state of your component.
    - A state variable scope is inside the component.
4. How do we create Super-powerful variable ?
    - for that we use `'React Hooks'`.
5. Difference between normal Javascript variable and state variable

```
* Local State Variable - Super Powerful Variable
    const [listOfRestaurants] = useState();

* Normal Javascript Variable
    let listOfRestaurants;
```

Introducing React-Hooks

1. **What is Hook ?**
    - It's simply a regular JavaScript function.
    - Which is a Utility functions.
    - it gives utility function such as `useState()`, `useEffect()` etc.
    - However, it becomes powerful when used within React, as it's provided to us by React itself. These pre-built functions have underlying logic developed by React developers. When we install React via npm, we gain access to these superpowers.

or

Two crucial hooks we frequently utilize are:

```
+ useState()
+ useEffect()
```

2. `useState()`

- First, we have to import as a named import from 'react'.
- We are using `useState()` inside the body component to create a `state variable`.

  Look at the syntax below.

  For import :

  ```
  import { useState } from "react";
  ```

  For using:

  ```
  // syntax of useState()

  const [listOfRestaurant , setListOfRestaurant] = useState
  ([]);
  ```

- In the provided code, we pass an empty array `[]` as the initial value inside the `useState([])` method. This empty array serves as the default value for the `listOfRestaurant` variable.
- If we pass the `listOfRestaurant` were we stored all the restaurant data inside the `useState()` as the default data, it will render the restaurants on the screen using that initial data.
- we can also write `useState()` in a different way :
- it is just an destructuring

  ```
  const arr = useState(resList);
  ```

- **first way :**

  ```
  const [restaurantsList, setRestaurantsList] = arr;
  ```

- **second way :**

```
const restaurantsList = arr[0];
const setRestaurantsList = arr[1];
```

3. **How could we modify the list of restaurant ?**
    - To modify, we use the second argument `setListOfRestaurant` we can name as we wish. `setListOfRestaurant` used to update the list.
        - **Why can't we just modify it directly?**
            - Because there needs to be a trigger to start the `Diff` algorithm and update the UI.
            - That's why they created the second function.
            - whenever we call the second function, it will automatically be render your component.

4. **How can we display filtered restaurants dynamically on UI(display screen)? I am repeating the same question which we had previously.**
    - We are using `setListOfRestaurant` inside the call-back function to show the filtered restaurant.
    - On clicking the button we will see the Updated top-rated restaurant.

```
<button
    type="button"
    className="btn btn-primary"

    onClick={() => {
        const filteredList = listOfRestaurants.filter(
            (res) => res.info.avgRating > 4.2
        );
        setRestaurantsList(filteredList);
    }}
>
    Top Rated Restaurant
</button>
```

> **Whenever a state variable updates, react will re-render the component. That logic of updating this UI is known as `re-rendering.`**

> 📢 NOTE:

```
- The crucial point about State variables is that
whenever they update, React triggers a reconciliation
cycle and re-renders the component.
- This means that as soon as the data layer changes,
React promptly updates the UI layer. The data layer is
always kept in sync with the UI layer.
- To achieve this rapid operation, React employs a
```

```
reconciliation algorithm, also known as the diffing
algorithm or React-Fibre which we will delve into further
below.
```

5. **React is often praised for its speed, have you ever wondered why?** 🤔

    - At the core lies `React-Fiber` - a powerhouse reimplementation of React's algorithm. The goal of React Fiber is to increase its suitability for areas like animation, layout, and gestures. Its headline feature is incremental rendering: the ability to split rendering work into chunks and spread it out over multiple frames.
    - These days, we can use *`JavaScript`* and *`React`* alongside popular libraries like *`GSAP`* (GreenSock Animation Platform) and *`Three.js`*.
    - These tools allow us to create animations and 3D designs using the capabilities of JavaScript and React.

    But how does it all work behind the scenes?

    - When you create elements in React, you're actually creating `virtual DOM objects`. These virtual replicas are synced with the `real DOM`, a process known as `"Reconciliation"` or the React `"diffing"` algorithm.
    - Essentially, every rendering cycle compares the new UI blueprint (updated VDOM) with the old one (previous VDOM) and makes precise changes to the actual DOM accordingly.

    It's important to understand these fundamentals in order to unlock a world of possibilities for front-end developers!

    Do you want to understand and dive deep into it?

    Take a look at this awesome React Fiber architecture repository on the web: https://github.com/acdlite/react-fiber-architecture

6. **How do you build a large scale application? what makes it super fast?**

    - React will make these DOM operations super fast & efficient.

7. **Why react is fast?**

    - It has a virtual DOM
    - It has a Diff algorithm which is very efficient.
    - It can do efficient DOM manipulation.
    - It can find out that Diff and update the UI.

8. **Notes : -**

    - React uses something known as `Reconciliation` algorithm.
    - This reconciliation algorithm is also known as react fibre.
    - Diff algorithm basically finds out the difference between the `updated virtual DOM` and the `previous virtual DOM`.

        - What it will do ?
            - It will then calculate the difference and then actually update the DOM on every render cycle.

    - This React Fibre is a new way of finding the diff and updating the DOM. that's how React is Faster.