CRA

What does CRA stand for?

"CRA" stands for "Create React App". It is a command-line interface tool that helps developers create new React projects with minimal setup and configuration.

Create React App sets up a development environment that includes all the necessary tools and libraries needed to build a modern React application. This includes a development server, webpack configuration, Babel for transpiling code, and more.

Using CRA allows developers to quickly get started with React development without having to spend time setting up a build system or configuring a project from scratch. It is a popular tool in the React community and is recommended for new React developers who want to get up and running quickly.

Why do we need CRA?

CRA (Create React App) is a popular tool used in the React development community to bootstrap new React projects. It provides a pre-configured setup for developing React applications, which helps to eliminate the need for developers to manually configure a new project from scratch.

Here are some reasons why CRA is useful in React development:

1. Faster Setup: CRA provides a pre-configured setup for developing React applications, which helps developers get started with a new project quickly.
2. Best Practices: CRA follows best practices for building React applications, which helps ensure that the application is structured and configured correctly.
3. Easy to Use: CRA is designed to be easy to use and understand, even for developers who are new to React development.
4. Testing: CRA includes pre-configured testing tools, such as Jest and Enzyme, which make it easier to write and run tests for the React application.
5. Scalability: CRA provides a scalable architecture that can be easily extended and customized as the project grows and evolves.

Overall, CRA simplifies the development process, reduces development time, and helps ensure that React applications are built according to best practices.

What are the advantages of CRA?

1. **Quick and Easy Setup:** CRA provides a pre-configured setup for developing React applications, which helps developers to start a new project quickly and easily without having to spend time on setting up the project from scratch.
2. **Pre-Configured Tooling**: CRA includes a pre-configured set of tooling for building and testing React applications, including Webpack, Babel, Jest, and Enzyme. This means that developers don't have to worry about configuring and integrating these tools manually.
3. **Consistent and Optimised Performance**: CRA uses a set of best practices for building React applications, such as code splitting and caching, which helps to ensure that the application runs consistently and efficiently across different devices and platforms.
4. **Easy to Maintain**: CRA provides a well-organized and structured codebase that makes it easy to maintain and update the application as it evolves over time.
5. **Community Support**: CRA has a large and active community of developers who contribute to its development and provide support to other developers. This means that developers can benefit from the collective knowledge and experience of the community when using CRA.

Overall, using CRA can help developers to streamline their development process, reduce development time, and build high-quality React applications that are optimised for performance and easy to maintain.

How to create React Application using CRA

Creating a React application using CRA (Create React App) is a straightforward process that involves a few simple steps:

1. **Install Node.js**: CRA requires Node.js to be installed on your machine. You can download and install it from the official website.
2. **Install Create React App**: Open your terminal or command prompt and run the following command to install Create React App globally:

npm install -g create-react-app

3. **Create a new React App**: Once you have installed Create React App, you can create a new React application by running the following command in your terminal or command prompt:

npx create-react-app my-app

Replace **my-app** with the name you want to give your application.

4. **Start the Application**: Once the application has been created, navigate into the project directory by running the following command:

```
cd my-app
```

Then start the application by running the following command:

```
npm start
```

This will start the development server and open the application in your default browser. You can now start developing your React application using CRA.

Note: CRA also provides many other options for configuring and customising your React application. You can refer to the official documentation for more information on these options.

React folder structure :

Folder Structure:

The folder structure of a React application typically includes a number of files and directories that are organised according to best practices for React development. The most common

directory structure for a React application includes the following directories:

- **node_modules**: This directory contains all of the dependencies that your React application requires.
- **public**: This directory contains the static assets that are served by your application, such as HTML files, images, and other media files.
- **src**: This directory contains the source code for your React application, including the components, styles, and other resources.
- **package.json**: This file contains the metadata for your React application, including the dependencies and scripts that are required to run and build the application.

App Component:

The App component is the top-level component of a React application. It represents the root of the component tree and is responsible for rendering all of the other components in the application. The App component typically includes a render method that returns the JSX code for the component tree.

Application Flow:

In a React application, the application flow is determined by the hierarchy of components in the component tree. Each component is responsible for rendering its own part of the user interface and passing data down to its child components as props. When a component receives new props or state, it re-renders its own part of the user interface and propagates the changes down to its child components.

Start and Stop React App:

To start a React application, navigate to the project directory in your terminal or command prompt and run the following command:

```
npm start
```

This will start the development server and open the application in your default browser. To stop the application, you can press Ctrl + C in the terminal or command prompt.

Access React App in other Browsers:

By default, when you start a React application using npm start, it will open the application in your default browser. However, if you want to access the application in other browsers, you can do so by opening a new browser window or tab and navigating to the following URL:

```
http://localhost:3000
```

This will open the application in your browser, and you can interact with it as you would with any other web application.

Creating multiple components in react

Here are the steps to create multiple components in a React application:

1. **Create a new component file**: To create a new component in React, create a new file with a .js extension in the src directory of your project. For example, you could create a file called Header.js to define a header component for your application.
2. **Define the component**: In the new component file, define the component by creating a new JavaScript function or class. The function or class should return the JSX code that defines the component's user interface.
3. **Export the component**: To use the new component in other parts of your application, you need to export it from the component file. You can do this by adding the following line of code at the end of the file:

```
export default Header
```

Replace **Header** with the name of your component.

4. **Import the component**: Once the new component is defined and exported, you can import it into other parts of your application where you want to use it. To import the component, add the following line of code at the top of the file where you want to use it:

```
import Header from './Header'
```

Replace **Header** with the name of your component file.

5. **Use the component**: Once the component is imported, you can use it in your JSX code by rendering it as a component. For example, you could render the Header component in the render method of another component as follows:

```
render() {
  return (
    <div>
      <Header />
      // other JSX code
    </div>
  );
}
```

This will render the **Header** component in your user interface, along with any other JSX code that you include in the render method.

By following these steps, you can create multiple components in your React application and use them to build a rich and dynamic user interface.

Parent Child Relationship Between Components in react:

In React, components can be organised in a parent-child relationship where a parent component can pass data to its child component through props. The child component can then use this data to render its own user interface or pass it down to its own child components.

Here's an example of a parent component that renders a child component:

```
function Parent() {
  const greeting = "Hello, world!";

  return (
    <div>
```

```
      <Child greeting={greeting} />
    </div>
  );
}
```

In this example, the Parent component is defined as a function that returns some **JSX** code that renders a Child component. The Parent component also defines a greeting variable that is passed to the Child component as a prop.

Here's an example of a child component that receives props from its parent:

```
function Child(props) {
  return (
    <div>
      <p>{props.greeting}</p>
    </div>
  );
}
```

In this example, **the Child component** is defined as a function that receives props as its argument. The Child component then

uses the greeting prop that was passed down from the Parent component to render a **<p> tag** with the text **"Hello, world!".**

This parent-child relationship can be nested to create a hierarchy of components. In this hierarchy, the parent component can pass data to its child components, which can then pass data down to their own child components.

For example, consider the following component hierarchy:

```
<App>
  <Header />
  <Main>
    <Section1 />
    <Section2 />
  </Main>
  <Footer />
</App>
```

In this hierarchy, the **App** component is the parent component that renders four child components: **Header**, **Main, Section1, Section2, and Footer.** The Main component is itself a parent component that renders two child components: **Section1 and Section2.**

Through the use of props, each child component can receive data from its parent component and use it to render its own

user interface or pass it down to its own child components. This allows for a highly organised and modular code structure in a React application.

Integrating Css With React using External Styling and Inline Styling in react :

In React, you can apply CSS styles to your components using external style sheets or inline styles.

External Styling

One way to apply CSS styles to your React components is by using external style sheets. To do this, you can create a separate CSS file and import it into your component. Here's an example:

1. Create a new CSS file styles.css and add some styles:

```
.text {
  color: blue;
  font-size: 16px;
}
```

2. Import the **styles.css** file into your component:

```
import React from "react";
import "./styles.css";

function MyComponent() {
  return (
    <div>
      <p className="text">Hello,
world!</p>
    </div>
  );
}

export default MyComponent;
```

In this example, we import the styles.css file using import **"./styles.css";**. Then, we use the className prop to apply the

text class to our **<p>** tag. This will apply the blue colour and **16px** font size styles to our text.

Inline Styling

Another way to apply CSS styles to your React components is by using inline styles. To do this, you can use the style prop to add styles directly to your elements. Here's an example:

```
import        from "react"

function MyComponent
  const
            "blue"
               "16px"


  return
    <div>
      <p style={styles}>
      </p>
    </div>
```

```
export default
```

In this example, we define a styles object with the **colour** and **fontSize** properties set to blue and **16px** respectively. Then, we use the **style** prop to apply the **styles** object to our **<p>** tag. This will apply the blue **colour** and 16px font size styles to our text.

Using inline styles can be useful when you need to apply dynamic styles based on user interaction or component state. However, it's generally recommended to use external style sheets for more complex or reusable styles.

SOME INTERVIEW QUESTIONS 👍

1. **What is React and how does it differ from other front-end frameworks/libraries?**

2. **What are the advantages of using Create React App (CRA) for building React applications?**

3. What is the difference between external and inline styling in React? When would you use one over the other?

4. What is a React component and how would you create one?

5. What is the parent-child relationship in React and how can data be passed between them?

6. How would you add event handling to a React component?