**React JS Tutorial**

Contents

[Reference 4](#_Toc143767205)

[ReactJS My GitHub 4](#_Toc143767206)

[Section 1: Getting Started 5](#_Toc143767207)

[1. About ReactJS 5](#_Toc143767208)

[Section 2: JavaScript Refresher 6](#_Toc143767209)

[2. The Spread and Rest Operator 6](#_Toc143767210)

[3. De structuring 6](#_Toc143767211)

[4. References and Primitive Types 7](#_Toc143767212)

[5. Array Function 7](#_Toc143767213)

[Section 3: React Basic & Working with Components 8](#_Toc143767214)

[6. Module Introduction 8](#_Toc143767215)

[7. What are Components & why React is all about them. 8](#_Toc143767216)

[8. React code is written in declarative way! 10](#_Toc143767217)

[9. Create new React project. 11](#_Toc143767218)

[10. Analysing the standard React Project 12](#_Toc143767219)

[11. Introducing JSX 13](#_Toc143767220)

[12. How React Works 13](#_Toc143767221)

[13. Building the First Custom Component 13](#_Toc143767222)

[14. Writing More Complex JSX code 14](#_Toc143767223)

[15. Adding Basic CSS Styling 14](#_Toc143767224)

[CSS @media rules: 14](#_Toc143767225)

[16. Outputting Dynamic Data and Working with Expressions in JSX 15](#_Toc143767226)

[17. Passing data via props 16](#_Toc143767227)

[18. Alternative ways of passing & receiving / handing props 19](#_Toc143767228)

[Pass entire object 19](#_Toc143767229)

[Object destructuring 20](#_Toc143767230)

[19. Organizing the react components with expense tracker example. 21](#_Toc143767231)

[React component and children (props.children). 21](#_Toc143767232)

[Section 4: Practice components basics 23](#_Toc143767233)

[Section 5: React State & Working with Events 25](#_Toc143767234)

[20. Module Introduction 25](#_Toc143767235)

[21. Listening to Events and Working with Event Handlers. 25](#_Toc143767236)

[22. How Components Functions Are Executed 25](#_Toc143767237)

[23. Working with State 25](#_Toc143767238)

[24. Adding Form Inputs 28](#_Toc143767239)

[Other Topics 29](#_Toc143767240)

[25. Using bootstrap in React 29](#_Toc143767241)

[Appendix 29](#_Toc143767242)

[TO DO 29](#_Toc143767243)

# Reference

|  |  |
| --- | --- |
| **Reference** | **URL** |
| Tutorial | [React Documentation](https://legacy.reactjs.org/docs/getting-started.html) |
| Udemy Course | [React - The Complete Guide 2023 (incl. React Router & Redux)](https://www.udemy.com/course/react-the-complete-guide-incl-redux/?kw=React+the+complete&src=sac) |
| [Udemy GitHub](https://github.com/academind/react-complete-guide-code) |
| Coursera | [meta-front-end-developer](https://www.coursera.org/professional-certificates/meta-front-end-developer) |

# ReactJS My GitHub

|  |  |
| --- | --- |
| **Item** | **URL** |
| GitHub URL | <https://github.com/Mansoor-R/ReactJs.git> |
|  |  |
|  |  |

# Section 1: Getting Started

### About ReactJS

ReactJS is an open-source JavaScript library used for building user interfaces (UI) or front-end components. It was developed and is maintained by Facebook, as well as a community of individual developers. React allows developers to create reusable UI components and efficiently manage the state of these components. It is one of the most popular libraries for front-end development and is widely used in web application development.

Key features and concepts of ReactJS:

**Components**: React organizes the user interface into reusable building blocks called components. Components encapsulate the UI and its behaviour, making it easier to maintain and reuse code.

**Virtual DOM**: React uses a Virtual DOM to efficiently update the user interface. When there is a change in the state of a component, React creates a virtual representation of the DOM, compares it with the previous version, and then applies the necessary changes to the actual DOM, reducing the need for direct DOM manipulation.

**JSX**: JSX (**JavaScript XML**) is an extension to JavaScript that allows developers to write HTML-like syntax within JavaScript code. It enables a declarative way of defining UI components.

**State and Props**: React components can have two types of data: state and props. State represents the internal data of a component that can change over time, while props are the properties passed from a parent component to its children.

**One-Way Data Flow**: React follows a one-way data flow where data flows from parent components to child components. This unidirectional data flow ensures predictable and manageable data updates.

**React Hooks**: Introduced in React 16.8, hooks are functions that allow developers to use state and other React features without writing a class. Hooks provide a more straightforward way to manage component state and lifecycle.

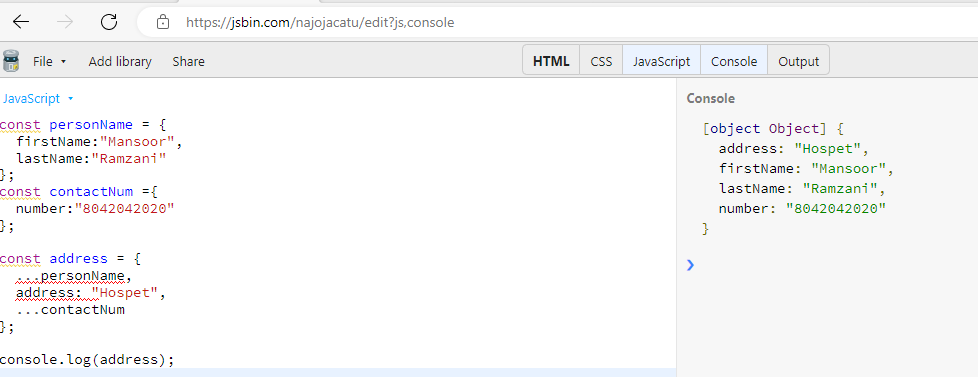
**Lifecycle Methods**: React components have various lifecycle methods that allow developers to perform actions at different stages of the component's existence, such as mounting, updating, and unmounting.

**Context**: React Context allows data to be passed through the component tree without the need to pass props explicitly at every level. It is useful for managing global state or sharing data that is required by many components.

# Section 2: JavaScript Refresher

Resource: <https://github.com/academind/react-complete-guide-code/tree/03-react-basics-working-with-components>

### The Spread and Rest Operator



### De structuring

Easily extract array elements or object properties and store them on variables.

|  |
| --- |
| **Array De-structuring** |
| [a, b] = [‘Hello’, ’Mansoor’] |
| Console.log(a) // Hello  Console.log(b) // Mansoor |
|  |

|  |
| --- |
| **Object De-structuring** |
| {name} = [name: ‘Mansoor’, age: 28] |
| Console.log(name) // Mansoor  Console.log(age) // undefined |

### References and Primitive Types

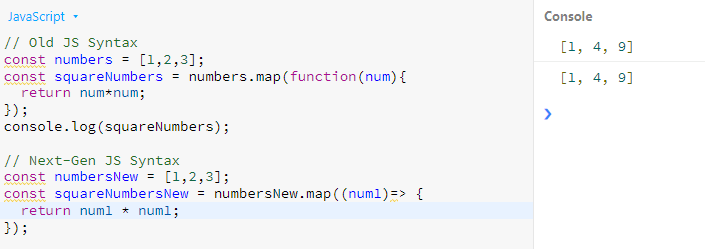
Graphical user interface, text

Description automatically generated with medium confidence

Text

Description automatically generated

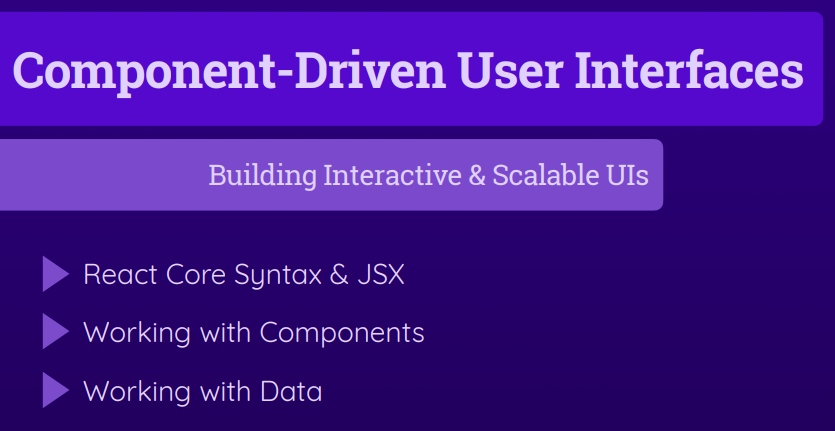
### Array Function



# Section 3: React Basic & Working with Components

### Module Introduction

We are going to learn following concepts along with examples.



### What are Components & why React is all about them.

React makes building complex, interactive and reactive user interface simpler.

Components are reusable building blocks of HTML, CSS, or JavaScript. Components allow reusability and separation of Concerns.





A blue symbol with yellow text

Description automatically generated

A blue and yellow symbol with a circle

Description automatically generated

A blue and yellow symbol with a circle

Description automatically generated

A blue and yellow symbol with white text

Description automatically generated

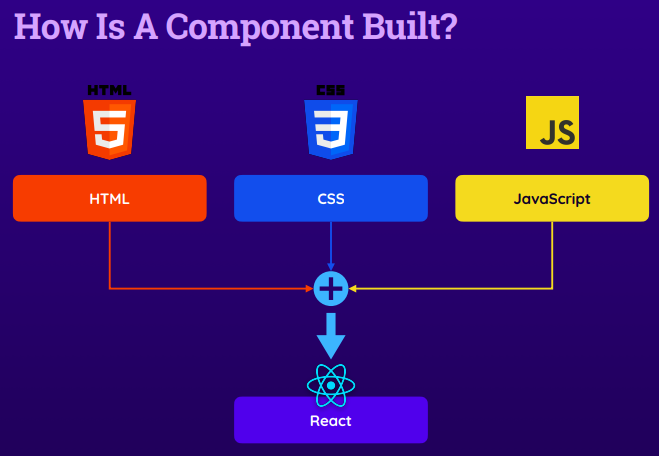
A screenshot of a computer

Description automatically generated

### React code is written in declarative way!

React allow you to create re-usable and interactive components consists of HTML, JavaScript (and CSS).

**Declarative** – Define all the desired state(s) and let React figure out the actual JavaScript DOM instruction. (You declare HTML and let React render it on DOM unlike Vanilla JavaScript where you define DOM).



A screenshot of a computer

Description automatically generated

### Create new React project.

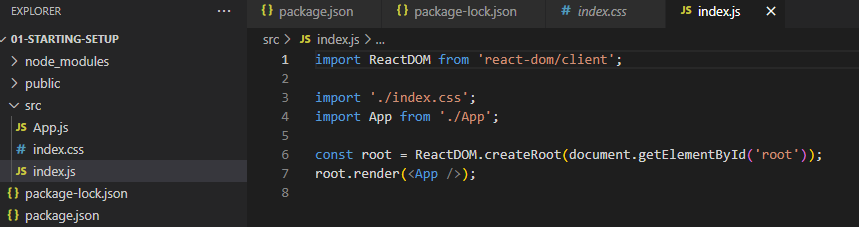
Create-react-app is used to create new React application this command requires NodeJS. So install NodeJS before creating the react project.

|  |
| --- |
| **Syntax**:  npx create-react-app my-app // Create React project  cd my-app // change directory  npm start // start the application on local development environment  URL: http:localhost/3000  CTRL + C to stop the local server |

Note: Create project name can’t contain capital letters.

### Analysing the standard React Project

Index.js is the first executable file here…



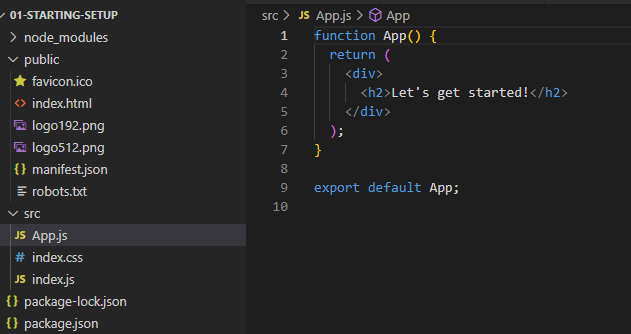
Index.html is the single html file in the project. Observe the div with id=”root” which does not hold any content but to this div react populate the UI. The same root div is selected in the react code above. On the root object the app is rendered.

Now what is that <App />. In the port we import app.js.

Text

Description automatically generated

In App.js function, a html code is returned.



### Introducing JSX

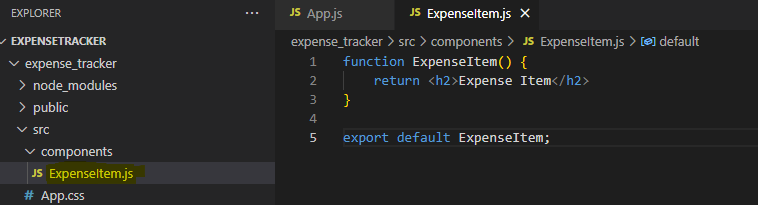
The code that we write is transformed behind the screen.

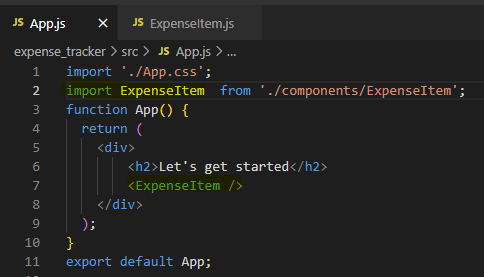
### How React Works

### Building the First Custom Component

1. Build one component in one file under components folder
2. File name must tell the logic, always start with capital letter and capital letter for each of the words without space.
3. Lower case compounds are basic html components and Uppercase (First charact and first chart of following words) components are custom components.
4. A component in React is just a JavaScript function. It returns JSX code.
5. In each component there must be only one root element.

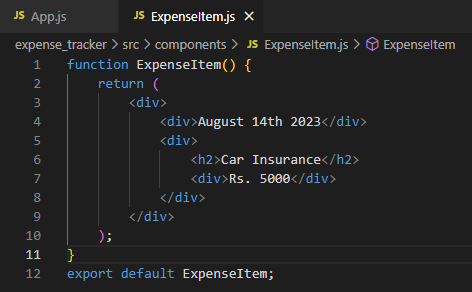
Ex: In the below example new custom components “ExpenseItem” is created.





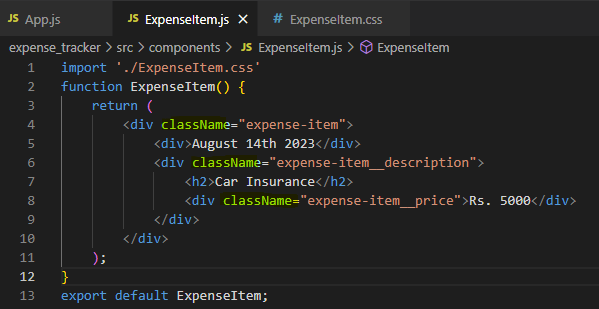
### Writing More Complex JSX code

1. CSS files need to be imported using import command.
2. Key word class need to replace with className.
3. It is standard not to put complex JavaScript code in JSX rather write this code outside.
4. In each component there must be only one root element.



### Adding Basic CSS Styling

* CSS file need to be imported using import command.
* Keywork class is replaced with className.



#### CSS @media rules:

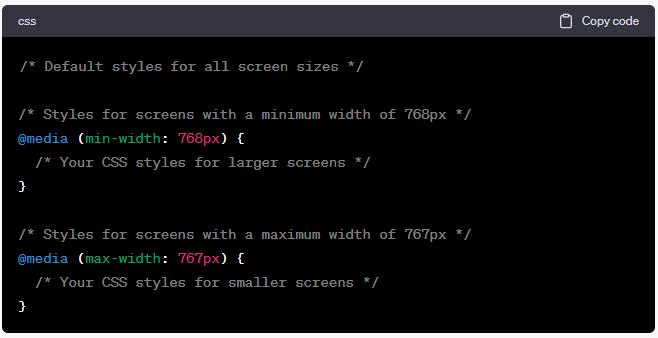
In CSS, the @media rule is used to apply different styles to a web page based on the characteristics of the device or viewport that the page is being displayed on. This allows you to create responsive designs that adapt to various screen sizes and orientations.

The **min-width** and **max-width** are commonly used properties within the @media rule to define specific conditions for applying styles. Here's what they mean:

**min-width**: This property specifies the minimum width of the viewport at which the styles defined within the @media rule will be applied. In other words, if the viewport width is equal to or greater than the specified min-width value, the styles will take effect.

**max-width**: This property specifies the maximum width of the viewport at which the styles defined within the @media rule will be applied. If the viewport width is equal to or less than the specified max-width value, the styles will be applied.

Here's an example of how you might use the @media rule with min-width and max-width to create responsive styles for different screen sizes:



### Outputting Dynamic Data and Working with Expressions in JSX

* {} can be used to evaluate JavaScript expression during compilation.
* Using $ for Variables: Sometimes, developers use the dollar sign as a prefix for variable names to indicate that the variable holds a reference to a DOM element or a result of a jQuery selection.



Figure : Dynamic Data & CSS class

### Passing data via props

In React, "props" (short for properties) are **a mechanism for passing data from a parent component to its child components.** Props are used to provide information or configuration to a component so that it can render with specific data or behaviour. They are immutable, meaning they cannot be modified within the component that receives them; instead, they are controlled by the parent component that passes them.

Props allow components to be dynamic and reusable, as different instances of a component can receive different data through props, resulting in varied rendering based on the input data.

Ex: Passing single child components with single prop.

|  |  |
| --- | --- |
| Parent Component |  |
| Child Component |  |
| Output |  |

Ex: Passing multiple child components with multiple props.

|  |  |
| --- | --- |
| Parent Component |  |
| Child Component |  |
| Output |  |

### Alternative ways of passing & receiving / handing props

There are alternative ways of passing props from parent components to child following are few.

Pass entire object: Instead of passing individual properties the entire data frame can be passed, item object in this case.

|  |  |
| --- | --- |
| Parent Component |  |
| Child Component |  |
| Output |  |

Object destructuring:

Object destructuring is a JavaScript feature that allows you to extract the values of specific properties from an object into variables. This can be useful for making your code more concise and readable, especially when you are working with complex objects.

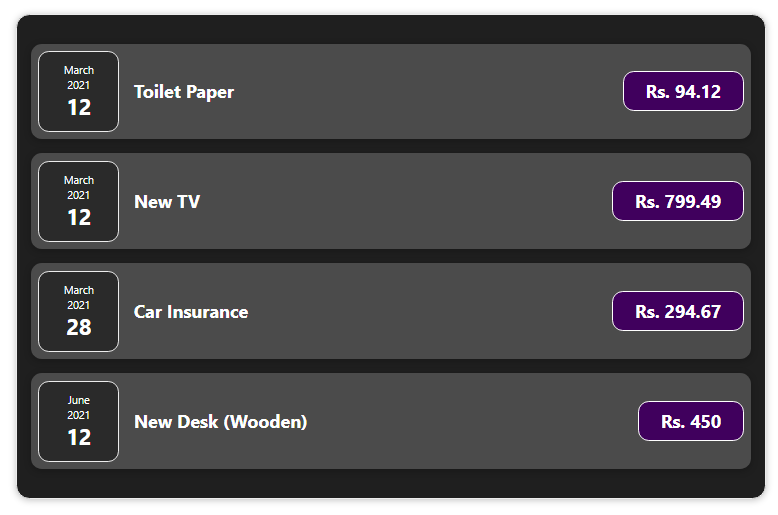
In React, object destructuring can be used to destructuring props that are passed to a component. This can make it easier to access the data in the props, and it can also help to make the code more readable.

|  |  |
| --- | --- |
| Parent Component |  |
| Child Component |  |
| Output |  |

### Organizing the react components with expense tracker example.

In this section we will learn about organizing the react components so that each components become manageable and reusable.

Ex: Expense tracker app as shown below.



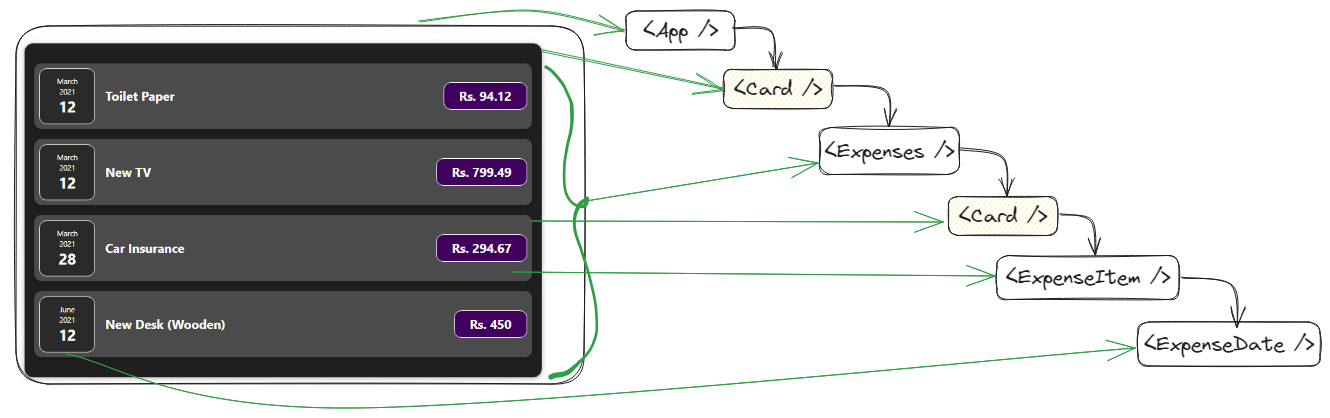


Figure : Expense Tracker Component Hierarchy 1

**Source code**:

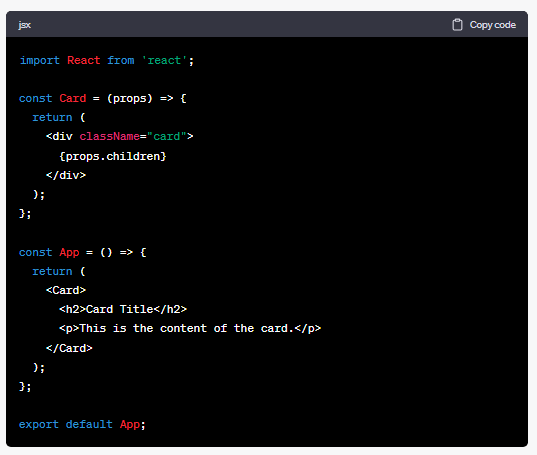
1. ReactJs\Applications\ExpenseTracker\expense\_tracker
2. [react-complete-guide-code/code/11-finished at 03-react-basics-working-with-components · academind/react-complete-guide-code (github.com)](https://github.com/academind/react-complete-guide-code/tree/03-react-basics-working-with-components/code/11-finished)

#### React component and children (props.children).

In React, props.children is a special prop that allows you to pass components, elements, or content between the opening and closing tags of a component. It provides a way to embed arbitrary JSX elements or components within another component, enabling the creation of more flexible and reusable components.

When a component has props.children, it means that any content placed between the opening and closing tags of that component when it's used in JSX will be passed to the component as the value of props.children.

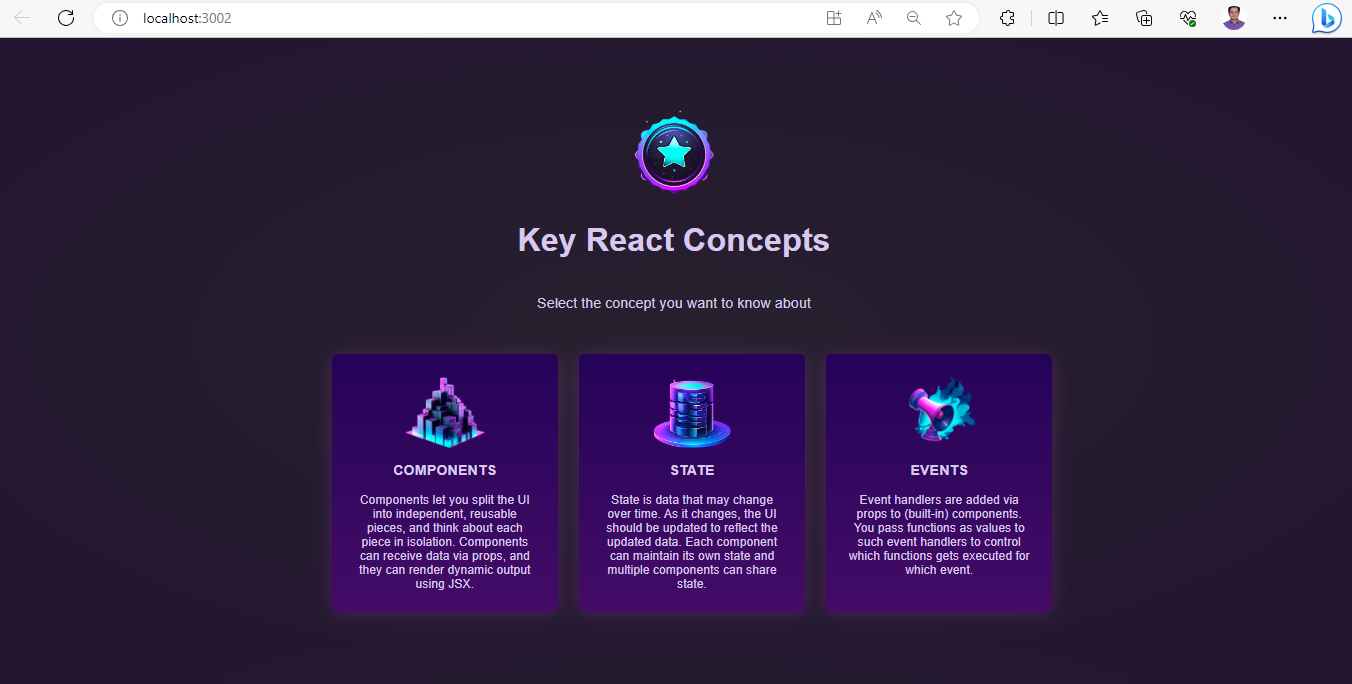
Here's a simple example to illustrate the concept:



In this example, the Card component takes props.children and renders it inside a <div> element with the class name "card." When you use the Card component in the App component and place content within it (such as <h2> and <p> elements), that content becomes the value of props.children in the Card component.

# Section 4: Practice components basics

**Exercise**: Create a UI as shown below

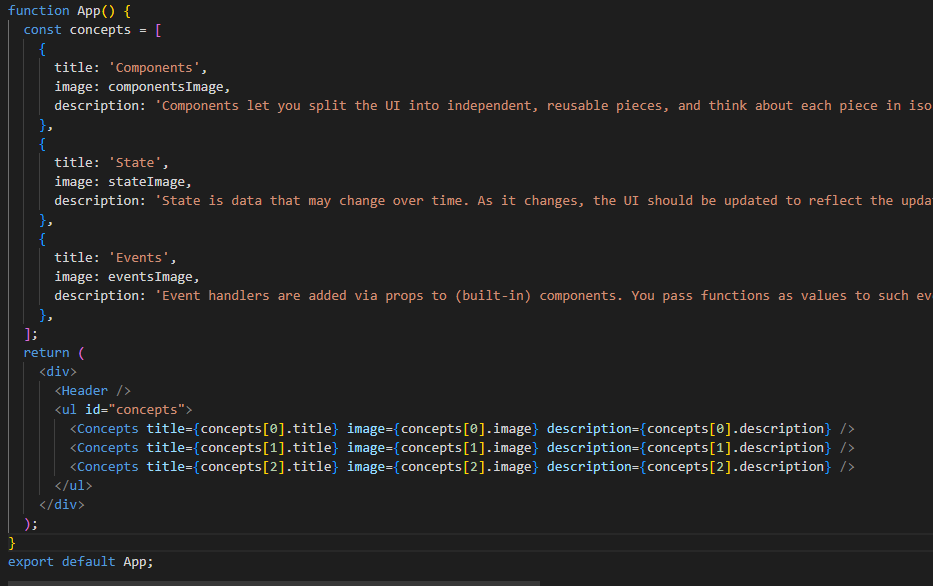


**Task**:

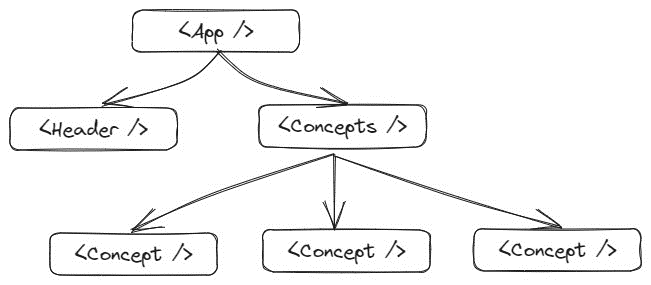
A blue rectangular with white text and a magnifying glass

Description automatically generated

Task 1: Output the key concepts data



Task 2: Identify possible components



Task 3: Create reusable custom components

A screen shot of a computer screen

Description automatically generated

Figure : Header Component

A screen shot of a computer program

Description automatically generated

Figure : Concepts components

Note: Source code is placed under “mygithub\ReactJs\Applications\ComponentBasic\componenet-basics”

# Section 5: React State & Working with Events

### Module Introduction

In this section we learn about user interactions and handling user events. Also, most important concept called state.

### Listening to Events and Working with Event Handlers.

* All the HTML controls can be added within JSX code, and all the events handlers are also can be used for these controls like onclick events etc.
* all the event listeners are exposed as property starting with on in React. For example, onClick.
* Event listeners calls functions. In the below example button click event listener call click Handle function.
* When parentheses are added to event listener function will get executed when DOM is loaded.

Text

Description automatically generated

### How Components Functions Are Executed

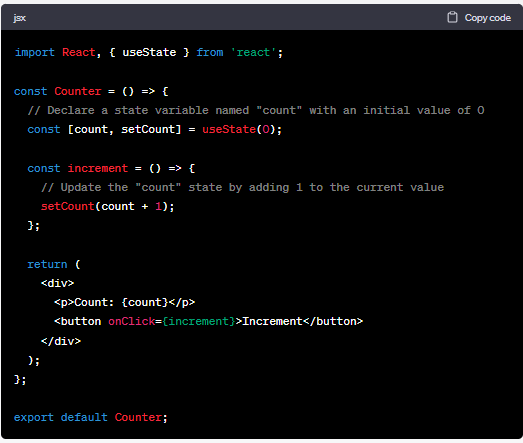
* React executes all the components one after the other serials. If a parent has nested component, then all the child components are executed first before going to next components. One component is not called 2nd time as call back.
* So now how do we update dome once page load is complete ?
* To update the state of any components once it is loaded React uses State.

### Working with State

In React, useState is a built-in hook that allows functional components to manage state. State represents data that can change over time and affects the rendering of a component. useState provides a way to declare and update state variables within a functional component.

Prior to the introduction of hooks, state management was primarily done using class components and the this.setState() method. Hooks, including useState, were introduced in React 16.8 to provide a more concise and intuitive way to manage state in functional components.

Here's a basic example of how to use useState in a React functional component:



In this example, the useState hook is used to declare a state variable named count with an initial value of 0. The hook returns an array with two elements: the current state value (count) and a function (setCount) that can be used to update the state value. When the "Increment" button is clicked, the increment function is called, updating the count state by adding 1 to its current value.

Key points about useState:

1. **Syntax**: const [state, setState] = useState(initialValue);

* **state**: The current state value.
* **setState**: A function used to update the state value.

1. **Immutable State**: React state is immutable. When you call setState, you provide the new value for the state variable, and React takes care of updating and re-rendering the component.
2. **Batching Updates**: React batches multiple state updates together to optimize rendering performance.
3. **Multiple State Variables**: You can use multiple useState hooks within a single component to manage different pieces of state independently.
4. **Functional Updates**: The setState function can also accept a function that calculates the new state based on the previous state. This is useful when the new state depends on the current state.

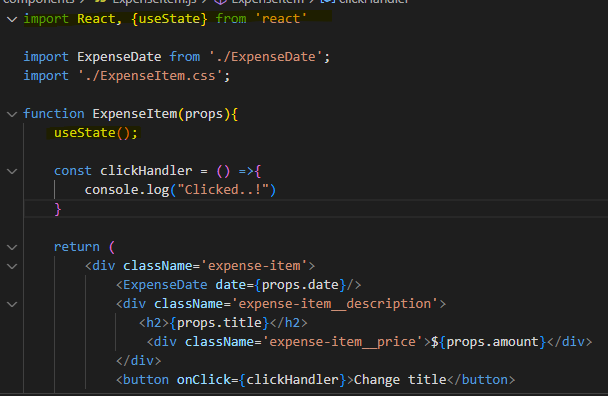
useState is just one of the many hooks available in React. Other hooks like **useEffect**, **useContext**, and **useReducer** provide additional functionality for managing side effects, context, and more. Hooks have greatly simplified the development of React components and made functional components a powerful alternative to class components.

**From Udemy course**:

To use state import useState library from React.

* useState is a react hook.
* This should be called inside react component function. And also they should not be called inside any nested function.
* useState has as specific format to follow as shown in 2nd screen.
* UseState set the values of variable title when function setTitle executed. And setTitle is executed on button click event.
* useState register state for specific instance meaning if an component is called multiple times then useState register state for specific instance.

|  |
| --- |
| function ConstructorFunction(prop) {  const [title, setTitle] = useState(prop.title); // Set Default value  setTitle(‘Updated’); // set new value.  } |



Text

Description automatically generated

* useState register state for specific instance meaning if an component is called multiple times then useState register state for specific instance.

### Adding Form Inputs

# Other Topics

### Using bootstrap in React

Refer : <https://blog.logrocket.com/using-bootstrap-with-react-tutorial-with-examples/#brief-introduction-javascript-css-frameworks>

# Appendix

|  |  |
| --- | --- |
| Js bin | [JS Bin - JS Bin](https://jsbin.com/najojacatu/edit?js,console) |
|  |  |

# TO DO

1. Dashboards
   1. Chart JS
   2. React integrate with Power BI

* By MS Power BI <https://www.youtube.com/watch?v=A5KFY5Jh1Uc>
* <https://github.com/amit-shuster/PowerBI-Developer-Videos/blob/master/Embedding%20Power%20BI%20in%20a%20React%20application/my-app/src/App.js>
* <https://www.npmjs.com/package/powerbi-client-react>

1. Calculator
2. Saving Plan