React

Single Page Application (SPA)

- a application that loads a single HTML page and dynamically updates that page as the user interacts with the app
- to develop SPAs,
    - we need to use a JavaScript framework or library
    - like
        - React
        - Angular
        - VueJs
- advantages
    - fast: similar performance to native apps
    - responsive: the app responds to user interactions (browser size changes),
        - to make the app responsive
            - we need to use CSS media queries
            - frameworks: bootstrap, tailwind
    - user-friendly

functional programming language

- function is considered as first class citizen
    - function is created as a variable of type function
- function can be passed as an argument to another function
- function can be returned from another function as return value
- map()
    - used to iterate over a collection to transform the values to new ones
    - accepts a function as a parameter which gets called every time for every value
    - the parameter function must return a transformed value for original value
    - all the transformed values will be returned a collection as a return value of map function
    - the size of returned collection is always same as original collection

```
// array of numbers
const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

// get squqre of each number
const squares = numbers.map((number) => number ** 2)
```

function reference

- a reference to a function
- a variable that holds a function body's address

```
// here the function1 is a function reference
// to the function body
function function1() {
  console.log('inside function1')
}
```

export and import

- export
    - used to export any entity from a file for others to import
    - a file can export multiple entities for others

    ```
    // App.jsx
    export function App() {
      ...
    }

    // main.jsx

    // importing with same name as that of the exported entity
    import {App} from './App.jsx'

    // importing with an alias
    import {App as MyApp} from './App.jsx'
    ```

- export default
    - by default only one entity (class, function, variable, constant) can be exported from a file with default keyword

    ```
    // App.jsx
    function App() {
      ...
    }
    export default App

    // main.jsx

    // importing with same name as that of exported entity
    import App from './App.jsx'

    // importing wth an alias
    import MyApp from './App.jsx'
    ```

React

- a JavaScript library for building user interfaces

React vs Angular

- React is a Library (developed in JS) and Angular is a Framework (developed in TypeScript)
- react development and performance is faster than angular
    - to make it faster, react uses a virtual DOM
    - it also has less memory consumption/footprint
- React has less learning curve than Angular
- React does not have any architecture whereas, Angular has a predefined architecture and tooling

important points

- do not use `class` as it is a reserved keyword in JavaScript, use `className` instead
- interpolation is done using `{}` in JSX
- interpolation always requires a scalar value and CAN NOT render an object

virtual DOM

- a lightweight copy of the real DOM (browser DOM) (document object)
- react uses virtual DOM to improve performance
- when we update the state of a component, react creates a new virtual DOM and compares it with the previous virtual DOM
- then it updates only the changed parts of the real DOM
- this process is called reconciliation

environment setup

- using CDN links
    - CDN: content delivery network
    - add react using CDN links

```
<html>
  <head>
    <!-- used for react development -->
    <script
      crossorigin
      src="https://unpkg.com/react@18/umd/react.development.js"
    ></script>

    <!-- used for react virtual dom development -->
    <script
      crossorigin
      src="https://unpkg.com/react-dom@18/umd/react-
dom.development.js"
```

```
    ></script>
  </head>

  <body>
    <div id="root"></div>
  </body>
</html>
```

- to create element use a function called `createElement`

```
// create element
// React is an object which will be used to create elements
// this object is provided by react library
(react.development.js)

// parameters
// 1st: name or type of the element (e.g. h1, h2 etc)
// 2nd: attributes or properties of the element (e.g. class, id,
style etc). this must be an object.
// 3rd: contents of the element (e.g. text, html etc)
const h2 = React.createElement('h2', {}, 'hello world')

// React 17 style of rendering an element
// get the root element
// this is the element where we will render our react elements
// const root = document.getElementById('root')

// render the element
// ReactDOM.render(h2, root)

// React 18 style of rendering an element

// create a root element
const root = ReactDOM.createRoot(document.getElementById('root'))

// render the element
root.render(h2)
```

- to create an element using JSX, use babel

```
<html>
  <head>
    <!-- used for react development -->
    <script
      crossorigin
      src="https://unpkg.com/react@18/umd/react.development.js"
    ></script>

    <!-- used for react virtual dom development -->
```

```
    <script
      crossorigin
      src="https://unpkg.com/react-dom@18/umd/react-
dom.development.js"
    ></script>

    <!-- babel compiler -->
    <script
src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
  </head>

  <body>
    <div id="root"></div>
    <script type="text/babel">
      const h2 = <h2>hello world</h2>
      const root =
ReactDOM.createRoot(document.getElementById('root'))
      root.render(h2)
    </script>
  </body>
</html>
```

- using package manager like vite

```
# install yarn on windows
> npm install -g yarn

# install yarn on linux or mac
> sudo npm install -g yarn

# create react application using vite
> npm create vite@latest <application name>
> yarn create vite <application name>

# go to the project directory
> cd <application name>

# install the dependencies
> npm install
> yarn

# start the application
> npm run dev
> yarn dev
```

    - project structure
        - node_modules

        - contains all the modules (dependencies) which are required to develop or run the application
        - will be downloaded every time when npm install or yarn install command is used
        - never commit this directory in your git repository
    - public
        - directory which contains public files
        - e.g. images, audio or video which are used in the application
    - src
        - directory which contains all the components of the application
        - contains
            - assets
                - directory which contains the assets (images, audio or video files)
            - main.jsx
                - contains the code to start react subsystem
                - contains the function createRoot(..).render()
            - index.css
                - contains global css rules which can be shared across the components in the application
            - App.jsx
                - contains the default component called as App
                - this is the startup component of every application
            - App.css
                - contains css rules need to applied on the App component
    - .gitignore
        - file which contains the rules of the files which needed to be committed in the git repository
    - eslint.config.js
        - contains configuration for eslint
        - lint is a program used to check the syntax of a selected language
    - index.html
        - only html file in the project which starts the application
    - package.json
        - contain the node configuration like name, dependencies or devDependencies etc.
    - vite.config.js
        - contains the vite configuration
    - yarn.lock
        - contains latest versions of the dependencies installed in the node_modules directory

react application startup

- vite will start a lite web server on port 5173
- the lite server starts loading index.html from the application
- index.html loads main.jsx file
- main.jsx calls createRoot() to create a root container to load react components
- and starts rendering first component named App
- App component start loading the user interface

component

- reusable entity which contains logic (in JS) and UI (in JSX)
- a component could as small as a part of an application
- or as big as an entire page
- types
    - class component
        - component created using a class
        - earlier (before react 16), class components were used for creating statefull component (component with state)
        - but after react 16 (in which react hooks were introduced), class components are not needed anymore
        - class components are having some overhead members compared to functional components
    - functional component
        - component created using a function
        - a javascript function which returns a JSX user interface
        - earlier (before react 16), functional components were used to create stateless components (components without state)
        - but with react 16 (react hooks), it is possible to store the state in a functional component
        - functiona components are preferred over class component
        - since the functional components do not have any overhead members, it is a way to create component compared to class component
- conventions
    - always start the component name with upper case
    - name of file should be same as the component name
    - if a component is reusable (like Person or Car), keep it in a directory named components
    - if a component is representing a page or screen, keep it in a directory named pages or screens
    - always use the props destructuring while defining the component
    - always keep one public component in a file

props

- is an object which is collection of all the properties passed to a component
- is the only way for a parent component to pass the data/properties to the child component
- props is a readonly object: the child component must not change the values sent by the parent component

```
function Person1(props) {
  return (
    <div>
      <div>name = {props['name']}</div>
      <div>address = {props['address']}</div>
    </div>
  )
}

function Person2(props) {
```

```
  const { name, address } = props
  return (
    <div>
      <div>name = {name}</div>
      <div>address = {address}</div>
    </div>
  )
}

function Person3({ name, address }) {
  return (
    <div>
      <div>name = {name}</div>
      <div>address = {address}</div>
    </div>
  )
}

function App() {
  return (
    <div>
      <Person1
        name='person1'
        address='pune'
      />
      <Person2
        name='person2'
        address='karad'
      />
      <Person3
        name='person3'
        address='satara'
      />
    </div>
  )
}
```

state

- object (collection of property-value pairs) maintained by compopent to trigger component re-render action
- if there is a change in the component state, the component will re-render itself
- unlike props, state object is readwritable
- every component will maintain its own state
- to add a state member inside a functional component use a react hook name useState()

state vs props

- state is read writable while props is read only
- when state changes, the component re-renders itself while, when props changes, component does not render itself

- state is maintained by individual component while, props will be sent by parent component to child component
- useState() hook is required to create state inside functional component while, no hook is required to send props to child component

react hook

- special function whihch starts with `use`
- types
    - built-in hooks
        - useState()
        - useEffect()
        - useReducer()
        - useMemo()
        - useId()
        - useRef()
        - useCallback()
        - useNavigate()
        - useLocation()
        - useParams()
        - useSelector()
        - useDispatch()
    - custom hooks
        - user defined hooks

useState()

- hook used to add a member inside a component's state
- accepts a parameter which is the initial value of the member
- returns an array with 2 values
    - 0th position: reference to the member (used to read the value from state)
    - 1st position: reference to the function to update the value in the state object

```
function Counter() {
  // create a state to store counter value
  const [couter, setCounter] = useState(0)
  return <div>counter: {counter}</div>
}
```

JSX

- a syntax extension for JavaScript
- allows you to write HTML code inside JavaScript
- how does it work?
    - babel is used to convert JSX code into JavaScript code

- babel is a JavaScript compiler

```
<script type="text/babel">
  // JSX code
  const h2 = <h2>hello world</h2>

  // babel converts the above code into the following code
  // const h2 = React.createElement('h2', {}, 'hello world')
</script>
```

VS extensions

- auto import:
  - https://marketplace.visualstudio.com/items/?itemName=NuclleaR.vscode-extension-auto-import
- auto tag renamer:
  - https://marketplace.visualstudio.com/items/?itemName=formulahendry.auto-rename-tag
- code snippets for react:
  - https://marketplace.visualstudio.com/items/?itemName=rodrigovallades.es7-react-js-snippets