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'
    ```

web storage

- web storage object is used to store collection key-value pairs
    - where key and value must be scalar value (string)
- the data will be persisted on client side (inside the browser)
- web storage is a browser specific object
- types
    - sessionStorage
        - used to store the data till the browser tab is open
        - when the tab is closed, the sessionStorage losses all the data
        - acts a temporory storage

```
// add or set a key-value in session storage
sessionStorage['username'] = 'user1'
sessionStorage.setItem('username', 'user1')

// get the value from session storage using its key
console.log(`username: ${sessionStorage['username']}`)
console.log(`username: ${sessionStorage.getItem('username')}`)

// remove the key from session storage
sessionStorage.removeItem('username')
```

    - localStorage
        - used to store the data in the form of key-value pairs
        - both the key and value must be scalar values (string)
        - the local storage will persist the data permananently (till user explicitly removes it)

```
// add or set a key-value in local storage
localStorage['username'] = 'user1'
localStorage.setItem('username', 'user1')

// get the value from local storage using its key
console.log(`username: ${localStorage['username']}`)
console.log(`username: ${localStorage.getItem('username')}`)

// remove the key from local storage
localStorage.removeItem('username')
```

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)
        - pages
            - directory which contains the application pages (screens)
        - components
            - directory which contains reusable components
        - services
            - directory which contains the services
            - a service file would contain the code to call the REST APIs
        - 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

component life cycle

- life cycle of component
- stages the component will pass from its creation to its death
- it provides the functionality for handling the life cycle event

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()
    - useContext()
    - 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>
}
```

useNavigate()

- react hook added by react-router-dom
- used to get the naigate() function reference
- navigate() function is used to perform dynamic navigation
- navigating from one to another component

```
import { useNavigate } from 'react-router-dom'

function Login() {
  // get navigate() function reference
  const navigate = useNavigate()

  const onLogin = () => {
    // check if user is successfully logged in
    navigate('/home')
  }

  return (
    <>
      <h1>Login</h1>
      <button onClick={onLogin}>login</button>
    </>
  )
}
```

useEffect()

- used to handle life cycle events of a component

- accepts 2 parameters

  - 1st parameter: callback function
  - 2ns parameter: dependency array

- event1: component is loaded (componentDidMount)

  - dependency array must be empty
  - the callback function gets called when the component gets mounted
  - it is similar to the constructor method in any class
  - it gets called only function

  ```
  function Home() {
    // this code here will handle the component did mount event
    useEffect(() => {
  ```

```
    // this function gets called immediately after component is
mounted
    console.log('Home component is mounted')
  }, [])

  return (
    <div>
      <h1>Home</h1>
    </div>
  )
}
```

- event2: component is unloaded (componentDidUnmount)
  - it gets called when the component gets removed from the screen
  - it gets called only once in its life cycle
  - dependency array must be empty
  - similar to destructor of any class

```
function Home() {
  // this code here will handle the component did mount event
  useEffect(() => {
    return () => {
      // this function gets called just before the component is
unloading from screen
      console.log('component is getting unloaded')
    }
  }, [])

  return (
    <div>
      <h1>Home</h1>
    </div>
  )
}
```

- event3: component state is changed
  - called as soon as the state of a component changes
  - this event gets fired irrespective of any state member
  - must NOT pass the dependency array

```
function Home() {
  const [n1, setN1] = useState(10)
  const [n2, setN2] = useState(20)

  const onUpdateN1 = () => {
    setN1(n1 + 1)
  }
  const onUpdateN2 = () => {
    setN2(n2 + 1)
  }

  useEffect(() => {
    // this function here will get called everytime when
    // the state changes
    console.log(`state changed..`)
  })

  return (
    <div>
      <Navbar />
      <div className='container'>
        <h1 className='page-header'>Dummy</h1>

        <div>n1: {n1}</div>
        <div>
          <button
            onClick={onUpdateN1}
            className='btn btn-success'
          >
            Update N1
          </button>
        </div>
        <div>n2: {n2}</div>
        <div>
          <button
            onClick={onUpdateN2}
            className='btn btn-success'
          >
            Update N2
          </button>
        </div>
      </div>
    </div>
  )
}
```

- event4: component state is changed
    - called as soon as the state of a component changes because of a required dependency

```
function Home() {
  const [n1, setN1] = useState(10)
  const [n2, setN2] = useState(20)

  const onUpdateN1 = () => {
    setN1(n1 + 1)
  }
  const onUpdateN2 = () => {
    setN2(n2 + 1)
  }

  useEffect(() => {
    // this function gets called when n1 changes
    console.log(`n1 changed..: ${n1}`)
  }, [n1])

  useEffect(() => {
    // this function gets called when n1 changes
    console.log(`n2 changed..: ${n2}`)
  }, [n2])

  return (
    <div>
      <Navbar />
      <div className='container'>
        <h1 className='page-header'>Dummy</h1>

        <div>n1: {n1}</div>
        <div>
          <button
            onClick={onUpdateN1}
            className='btn btn-success'
          >
            Update N1
          </button>
        </div>
        <div>n2: {n2}</div>
        <div>
          <button
            onClick={onUpdateN2}
            className='btn btn-success'
          >
            Update N2
          </button>
        </div>
      </div>
    </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>
```

context api

- it is a built-in feature of react used to share data among multple components
- context
  - instance of Context component used to provide (share) data
  - to create a context use createContent() function
- to share the context with components use following syntax
  - ...

```
import { createContext, useState } from 'react'

// create a context to share the data
export const CounterContext = createContext()

function App() {
  const [counter, setCounter] = useState(0)
  return (
    <div>
      <h1>App Component</h1>
      <CounterContext value={{ couter, setCounter }}>
        <Counter1 />
        <Counter2 />
      </CounterContext>
    </div>
  )
}
```

- to use the contxt in the child components, use useContext() react hook

```
import { CounterContext } from '../App'

function Counter1() {
  // get the counter and setCounter from counter context created in
App.jsx
  const { counter, setCounter } = useContext(CounterContext)

  const onIncrement = () => setCounter(counter + 1)
  return (
    <div>
      <div>counter: {counter}</div>
      <div>
        <button onClick={onIncrement}>increment</button>{' '}
      </div>
    </div>
  )
}
```

- use case: to share simple values like
  - login status
  - themse used in the application

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

external libraries

```
# install any package
> npm install <package name>
> yarn add <package name>
```

- react toastify

  - used to show the toast message
  - https://www.npmjs.com/package/react-toastify
  - yarn add react-toastify

  ```
  import { ToastContainer } from 'react-toastify'

  function App() {
    return (
      <div>
        <ToastContainer />
      </div>
    )
  ```

```
}

// to show the toast messages
// import {toast} from 'react-toastify'

// toast.warn('this is warning message')
// toast.error('this is error message')
// toast.info('this is info message')
// toast.success('this is success message')
```

- axios
  - used to make API calls
  - https://www.npmjs.com/package/axios
  - yarn add axios

```
import axios from 'axios'

async function makePostCall(email, password) {
  try {
    const url = 'http://localhost:4000/user/login'
    const body = { email, password }
    const response = await axios.post(url, body)
    console.log(response.data)
  } catch (ex) {
    console.log('exception: ', ex)
  }
}

async function makePostCallWithToken(title, description, price) {
  try {
    const url = 'http://localhost:4000/property/'
    const body = { title, description, price }
    const token = sessionStorage.getItem('token')
    const response = await axios.post(url, body, {
      headers: { token },
    })
    console.log(response.data)
  } catch (ex) {
    console.log('exception: ', ex)
  }
}

async function makeGetCall() {
  try {
    const url = 'http://localhost:4000/property'
    const response = await axios.get(url)
    console.log(response.data)
  } catch (ex) {
    console.log('exception: ', ex)
  }
}

async function makeGetCallWithToken() {
  try {
    const url = 'http://localhost:4000/my'
    const token = sessionStorage.getItem('token')
    const response = await axios.get(url, {
      headers: { token },
    })
    console.log(response.data)
  } catch (ex) {
    console.log('exception: ', ex)
  }
}
```

- react-router
  - used to add routing feature in react application
  - routing is used to provide navigation from one to another component
  - https://reactrouter.com/
  - yarn add react-router-dom
  - route in express is mapping of
    - http method (get, post, put, delete, patch)
    - url path
    - callback function (handler)
  - route in react is mapping of
    - url path
    - component
  - BrowserRouter
    - router provided by library to implement the routing
    - to load the required comopnent by inspecting the url path

```
// step1: wrap <App /> inside BrowserRouter
// main.jsx
import { BrowserRouter } from 'react-router-dom'

// Wrap the App component inside the BrowserRouter
createRoot(document.getElementById('root')).render(
  <BrowserRouter>
    <App />
  </BrowserRouter>
)
```

```
// step2: define all the routes
// App.jsx
import { Routes, Route } from 'react-router-dom'
import Login from './pages/Login'
```

```
function App() {

  return <>
    <Routes>
      <Route path="login" element={<Login />}>
    </Routes>
  </>
}
```

- static navigation
  - navigation from one component to another without using JS code (logic)
  - navigation will be implemented by using JSX
  - use static navigation when the destination (component) needs to open without having any condition (criteria)

```
import { Link } from 'react-router-dom'

function Login() {
  return (
    <>
      <h2>Login</h2>
      <Link to='/register'>Register here</Link>
    </>
  )
}
```

- dynamic navigation
  - navigation performed using JS code
  - used dynamic navigation when the destination (componen) needs to open by validation some condition (criteria)

```
import { useNavigate } from 'react-router-dom'

function Login() {
  // get navigate() function reference
  const navigate = useNavigate()

  const onLogin = () => {
    // check if user is successfully logged in
    navigate('/home')
  }

  return (
    <>
      <h1>Login</h1>
      <button onClick={onLogin}>login</button>
    </>
  )
}
```

- dynamic navigation with data

```
import { useNavigate } from 'react-router-dom'

function Properties() {
  const [properties, setProperties] = useState([])

  // get navigate() function reference
  const navigate = useNavigate()

  const onDetails = (property) => {
    // check if user is successfully logged in
    navigate('/details', { state: property })
  }

  return (
    <>
      <h1>Properties</h1>
      {properties.map((property) => {
        return (
          <div>
            <div>{property['title']}</div>
            <button onClick={() =>
onDetails(property)}>details</button>
          </div>
        )
      })}
    </>
  )
}
```

- tanstack router
  - use to add routing feature in react application
  - https://tanstack.com/router/latest
- bootstrap-icons
  - used to add the icons in react application
  - yarn add bootstrap-icons
- react-bootstrap-icons
  - used to add the icons in react application
  - yarn add react-bootstrap-icons

- redux-toolkit
  - used to implement redux architecture in react application
  - redux provides global state management
  - global state management is implmented using a (global) store which will be accessible to all the components in the react application
  - components
    - store
      - a collection of slices
      - slice is a feature which is meant to store some data in the form of key-value pairs
    - action
      - is an event which can be fired to either read the contents of a slice or to update/modify the cotents of a slice
      - an action is responsible for changing the state of store
    - reducer
      - a function or event handler which gets invoked when an action is fired
      - an action uses reducer to update the store state
      - reducer contains the logic to update the store state
  - installation
    - yarn add @reduxjs/toolkit react-redux
  - implementation of redux in react application
    - step1: create an empty store in the application

```
// src/store.js

import { configureStore } from '@reduxjs/toolkit'

// create a store
export const store = configureStore({
  reducer: {},
})
```

    - step2: add the store in the react application

```
import { createRoot } from 'react-dom/client'
import './index.css'
import App from './App.jsx'
import { Provider } from 'react-redux'
import { store } from './store.js'

createRoot(document.getElementById('root')).render(
  <Provider store={store}>
```

```
    <App />
  </Provider>
)
```

- step3: create a slice

```
// features/counter.slice.js
import { createSlice } from '@reduxjs/toolkit'

// create a slice
const counterSlice = createSlice({
  // unique name to identify the slice inside the store
  name: 'counter',

  // state to be maintained by the slice
  initialState: {
    count: 0,
  },

  // collection of actions and their respective reducers
  reducers: {
    // incrementAction is the action
    // the function is the reducer
    incrementAction: (state) => {
      // update the state
      state.count += 1
    },
    decrementAction: (state) => {
      state.count -= 1
    },
  },
})

// export actions
export const { incrementAction, decrementAction } =
counterSlice.actions

// export the reducers
export default counterSlice.reducer
```

- step4: add the slice into the store

```
// src/store.js

import { configureStore } from '@reduxjs/toolkit'

// create a store
export const store = configureStore({
  reducer: {},
})
```

  - useSelector()
    - used to read the contents from store

```
import { useSelector } from 'react-redux'

// read the count value from store
const { count } = useSelector((store) => store.counter)
```

  - useDispatch()
    - used to update the store's state
    - returns a dispatch function reference which is used to update the store state

```
import { useDispatch } from 'react-redux'
import { incrementAction } from '../features/counter.slice'

function Counter() {
  // get the dispatch function reference
  const dispatch = useDispatch()

  // increment the count
  const onIncrement = () => {
    // send the increment action
    dispatch(incrementAction())
  }

  return (
    <div>
      <button onClick={onIncrement}>increment</button>
    </div>
  )
}
```

- moment
  - used to manipulate the date or time
  - yarn add moment