URL

- Uniform Resource Locator
- used find the resource on internet
- e.g.
    - http://mywebsite.com
    - https://google.co.in/index.html
    - http://localhost:4000/product
    - http://localhost:4000/product?category=test
    - https://apple.com/index.html#top
- components
    - scheme
        - defines the protocol to be used to communicate with server
        - its optional
        - if it is not specified, http is by default used
    - domain name or ip address
        - mandatory
        - sever's location
        - domain name gets converted into ip address using DNS server
    - port number
        - port number of the web server process
        - its optionally present
        - if it is absent, the port number will be used as per the scheme selected
            - e.g. http (80), https (443)
    - file name or path
        - name of the resource to be loaded
        - it is optionally present
        - if it is absent, server by default returns a resource with index name
            - e.g. index.html, index.htm, index.aspx, index.php
    - query string
        - input passed to the page
        - it is optionally present
        - if it is present, the format of query string must be
            - ?key1=value1&key2=value2
            - e.g. ?firstName=steve&lastName=jobs
    - hash component
        - also known as proxy component
        - used to link a section within the page
        - it is optionally present
        - e.g. top is used to go to the top of the page

browser fundamentals / architecture

- network component

    - used to send the request to the server
    - used to get the response from the server
- rendering component / engine
    - also known as a layout engine
    - used to render the HTML page including CSS (convert the html to JavaScript)
- javascript engine
    - the heart of any browser
    - used to execute the JavaScript code
- user interface component
    - used to display the user interface
- web-storage component
    - used to store the data in the browser
    - e.g. local storage, session storage, cookies

node package manager

- there are few package managers available
    - npm
        - by default comes with node.js
        - used to install the packages
        - used to manage the packages
        - used to create the package.json file
    - yarn
        - used to install the packages
        - used to manage the packages
        - installation

```
# install yarn
> npm install --global yarn
```

        - commands

```
# initialize the package.json file
> yarn init

# install the packages
# > yarn add <package-name>
> yarn add multer mysql2 jsonwebtoken

# install the packages from package.json file
> yarn install
```

```
> yarn
```

- pnpm
  - used to install the packages
  - used to manage the packages
  - installation

```
# install pnpm
> npm install --global pnpm
```

React

- a JS library used to develop Single Page Application
- Single Page Application
  - contains only one html page
  - gets loaded only once when user visits the website
  - once loaded, it sends the request to the server only to get the data
  - it executed only on the client side (inside a browser)
- react is used to develop client side applications
- developed by Facebook and open sourced for other developers
- features
  - has a component-driven architecture
  - used to developer SPA type applications
  - used virtual DOM for improving the application performance
  - it has eco-system: React Router, React Redux Toolkit, React Native

project setup

```
# setup a react project using vite
# > npm create vite@latest <application name>
> npm create vite@latest myapp

# setup a react project using yarn
# > yarn create vite <application name>
> yarn create vite myapp

# go to the directory
> cd myapp

# install the dependencies
> npm install
# or
> yarn install
```

```
> yarn

# run the application
> npm run dev
# or
> yarn dev

# test the application
> npm run test
# or
> yarn test
```

used CDN links

- downloading the react library every time the page is loaded
- to use the CDN links:

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

<!-- Don't use this in production: -->
<script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
```

- react.development.js
  - used for developing the react application
  - used to create react components
- react-dom.development.js
  - used to render the react components inside the browser
- babel.min.js
  - used to convert the JSX code into JS code

using application managers

- using vite

```
# create react app using vite
> npm create vite@latest myapp

# go to the directory
> cd myapp

# install the dependencies
> npm install
```

```
# run the application
> npm run dev

# visit the url: http://localhost:5173/
```

project structure

- node_modules
    - contains all the dependencies
    - don't modify this folder
    - gets created when you run the command: npm install or yarn
- public
    - contains the static files
    - used to store the images, fonts, audio, video files etc.
- src
    - assets
        - contains the assets (resources) like images, audio, video files
    - App.css
        - contains the css rules for the App component
    - App.jsx
        - contains the startup component named App
        - when the application starts, it loads this component
    - index.css
        - contains the global css rules
        - all the css rules which can be shared across all the components
    - main.jsx
        - contains the startup function to load the first component
    - screens or pages
        - contains the components which represent the pages
    - services
        - contains the functions which are used to connect to the backend
    - components
        - contains the reusable components
        - these components are shared across the pages
- .gitignore
    - used to ignore the files and folders which are not required to be pushed to the git repository
- eslint.config.js
    - contains the configuration about the ES Lint program
    - linter is a program to find out the syntax errors

- index.html
    - the only html file in the project
    - this file loads all the react components
    - this file contains a div with id root which is considered to be the host for all the react components
- package.json
    - contains the configuration about the react application
    - scripts
        - contains commands which can be used with npm or yarn
    - dependencies
        - contains list of modules which will be compiled and added to the deployment package
        - the packages mentioned here are required to run the application
    - devDependencies
        - contains a list of modules which will be needed to develop the application
        - these modules will NOT be compiled into the deployable package
        - these packages are NOT required to run the application
- vite.config.js
    - configuration file for vite application package manager

application flow

- when the application starts (after yarn dev), vite starts a lite web server on port 5173
- web server loads the index.html
- the index.html file loads the main.jsx script
- in the main.jsx
    - finds the div having an id root
    - creates a root with the div for rendering the React components
    - loads the default/startup component named App
    - renders the App component into the div with 'root' id

data binding

- using the variable value inside a html tag
- in react, it will be done using interpolation
    - used the {} brackets for loading the variables value inside the html tag

```
const myvar = 100
<h3>{myvar}</h3>
```

- to render a simple variable use interpolation

```
const myvar = 100
<h3>{myvar}</h3>
```

- to render an object, split the object into properties and use interpolation to render those properties

```
const car = {
  model: 'triber',
  company: 'renault',
  price: 10
}

<div>
  <div>model = {car['model']}</div>
  <div>company = {car.company}</div>
  <div>price = {car.price}</div>
</div>
```

```
const car = {
  model: 'triber',
  company: 'renault',
  price: 10
}

const {mode, company, price} = car

<div>
  <div>model = {model}</div>
  <div>company = {company}</div>
  <div>price = {price}</div>
</div>
```

- to render an array, use the map function to iterate over the array and use interpolation to render the properties of the object

```
const cars = [
  { model: 'triber', company: 'renault', price: 10 },
  { model: 'seltos', company: 'kia', price: 20 },
  { model: 'creta', company: 'hyundai', price: 30 },
]
```

```
const div = cars.map((car, index) => {
  return (
    <div key={index}>
      <div>model = {car['model']}</div>
      <div>company = {car.company}</div>
      <div>price = {car.price}</div>
    </div>
  )
})
```

component

- reusable entity which contains user interface defined in html code
- a component can be loaded using the component name as a tag (enclosed by < and >)
- types
  - functional component
    - component created using a function
    - before react 16, functional components were used only for stateless implementation (the component which does not require to maintain the state)
    - since the react 16 introduced a concept called as a react hooks, it is possible now to create functional components to store the state
    - hence the class components are not need anymore and by default we use a function to create component
    - a javascript function which returns a JSX code to create its user interface
  - class component
    - component create using a class
    - before react 16, class components were used to create stateful components (a component which can maintain its state)

props

- is an object containing all the properties sent by a parent component to a child component
- it is a readonly object (if child component modifies the props, the new values will NOT be available in the parent component)

```
export default function Person(props) {
  return (
    <div>
      <div>name = {props.name}</div>
      <div>address = {props.address}</div>
    </div>
  )
}

// ------ App.jsx ----

// the Person component will receive the name and address as props object
;<Person
```

```
  name='person1'
  address='pune'
/>
```

- props drilling
    - passing the props from parent to child component
    - if the child component is not directly under the parent component, then the props will be passed to all the intermediate components
    - this is called as props drilling
    - to avoid props drilling, we can use context api

```
function First() {
  const [count, setCount] = useState(10)

  return <>
    <Child count={count} setCount={setCount}></Child>
  </>
}

function Child({count, setCount}) {
  return <>
    <GrandChild count={count} setCount={setCount}></GrandChild>
  </>
}

function GrandChild({count, setCount}) {
  return <>
    <h2>count = {count}</h2>
    <button onClick={() => setCount(count+1)}>Update</button>
  </>
}
```

event handling

- to handle any event in react application, first define a function withing the required component
- specify the function as event handler in the required tag
- note: please make sure you are not using the function call while configuring the event handler
- react will always pass an argument of type SyntheticBaseEvent which is an object of respective event

```javascript
function App() {
  const onButtonClick = () => {
    alert('button clicked')
  }

  return (
```

```
    <div>
      <button onClick={onButtonClick}>click here</button>
    </div>
  )
}
```

- to get input from user
  - create change event handler and
  - configure it as change event handler of the required input

```
function App() {
  const onTextChange = (event) => {
    // get user input
    const userInput = event.target.value
    console.log(`user input = ${userInput}`)
  }

  return (
    <div>
      user name:
      <input
        type='text'
        onChange={onTextChange}
      />
    </div>
  )
}
```

react hooks

- special function which starts with 'use' prefix
- hook must be called within a functional component outside of any inner function of a component

```
function Login() {
  // it allowed to call useState() here as it is called outside of any
inner function
  const [email, setEmail] = useState('')
  const [password, setPassword] = useState('')

  const onLogin = () => {
    // this is going to raise an error as the useState() hook is called
inside an inner function
    // const [test, setTest] = useState('')
  }
}
```

- e.g.

    - react system hook:
        - useState(): used to create a state member
        - useEffect(): used to handle component life cycle
        - useReducer(): used to maintain state
        - useCallback(): used to handle communication from parent to child
        - useContext(): used to manage a shared context
        - useMemo(): used to manage memoic function
        - useRef(): used to get the native reference of an element
    - react router
        - useNavigate(): used to navigate from one to another component
        - useLocation(): used to send parameters from one to another component
    - react redux toolkit
        - useDispatch(): used to get dispatcher to dispatch an action
        - useSelector(): used to read global store

useState()

- a react hook, used to add a member to the state object
- returns an array having
    - position0: reference to the state member (for reading the value)
    - position1: function to update the state member
- accept the default value of the member

```
function App() {
  // add a value to state
  const [value, setValue] = useState(0)

  return <h1>value: {value}</h1>
}
```

useEffect()

- a react hook, used to handle the component life cycle
- accepts two parameters
    - param1: callback function
    - param2: array of dependencies
- lifecycle stage 1: (componentDidMount) when the component gets mounted
    - the useEffect dependency array is empty
    - this function gets called only once its life cycle

```
function Component1() {
  // this function will be called when the component gets mounted
  useEffect(() => {
    console.log('component mounted')
```

```
  }, [])

  return <h1>component1</h1>
}
```

- life cycle stage2: (componentDidUnmount) the component is unmounted
  - the dependency array is empty
  - this function gets called only once its life cycle

```
function Component1() {
  useEffect(() => {
    // this function will be called when the component is mounted
    console.log(`Component is mounted`)

    return () => {
      // this function will be called when the component is unmounted
      console.log(`Component is unmounted`)
    }
  }, [])
}
```

- life cycle stage3: (componentStateUpdated) the component state is changed
  - the dependency array is null (missing)

```
function Component1() {
  const [count, setCount] = useState(0)

  useEffect(() => {
    // this function will be called when there is change in state
    console.log(`component state is changed`)
  })
}
```

- life cycle stage4: component state is changed because of dependency
  - the dependency array has at least one member
  - when the dependency array has more than one members, then the function will be called when one of the members gets updated

```
function Component1() {
  const [count1, setCount1] = useState(0)
  const [count2, setCount2] = useState(0)

  useEffect(() => {
    console.log('count1 is changed')
  }, [count1])
```

```
  useEffect(() => {
    console.log('count2 is changed')
  }, [count2])

  useEffect(() => {
    console.log('count1 or count2 is changed')
  }, [count1, count2])
}
```

state

- maintained by individual component
- state of a component is not shared with any other components
- unlike props, state is both readable and writable
- if state of a component changes, the component re-renders the UI where the state members are used

context api

- in react application, context is the data / information to be shared with multiple components
- context api is provided by react (no external package is required)
- context must be created using createContext()
- context must be shared with Provider property of context

```
// App.jsx

// create an empty context
export const AuthContext = createContext()

function App() {
  // create the state to be shared with child components
  const [user, setUser] = useState(null)

  return (
    <>
      <AuthContext.Provider value={{ user, setUser }}>
        <FirstComponent />
        <SecondComponent />
      </AuthContext.Provider>

      {/* this component wont be able to access the AuthContext */}
      <ThirdComponent />
    </>
  )
}
```

- to use the context the component must use react hook named useContext

```jsx
// FirstComponent.jsx

function FirstComponent() {
  // get the user from AuthContext
  const { user } = useContext(AuthContext)

  return (
    <>
      <h2>{user}</h2>
    </>
  )
}
```

```jsx
// SecondComponent.jsx

function SecondComponent() {
  // get the user from AuthContext
  const { setUser } = useContext(AuthContext)

  return (
    <>
      <button
        onClick={() => {
          setUser('test user')
        }}
      >
        set user name
      </button>
    </>
  )
}
```

external packages

- react-toastify
    - used to show toast messages
    - installation

```
# install react-toastify
> yarn add react-toastify
```

    - usage

```jsx
import { ToastContainer, toast } from 'react-toastify'
import 'react-toastify/dist/ReactToastify.css'
```

```
function App() {
  const notify = () => toast('Wow so easy!')

  return (
    <div>
      <button onClick={notify}>Notify!</button>
      <ToastContainer />
    </div>
  )
}
```

external libraries

- react-toastify: used to show toasts
- react-router: used to add routing in the react application
- redux toolkit: used to manage the global store
- axios: used to consume REST apis
- bootstrap: used to decorate the UI with predefined classes

react-router

- used to add routing in the react application
- installation

```
# install react-router
> yarn add react-router-dom
```

- configuration

```
// main.jsx

import { BrowserRouter } from 'react-router-dom'

createRoot(document.getElementById('root')).render(
  <BrowserRouter>
    <App />
  </BrowserRouter>
)
```

```
// App.jsx
import { Routes, Route } from 'react-router-dom'

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

```
        <Routes>
          <Route
            path='login'
            element={<Login />}
          />
          <Route
            path='register'
            element={<Register />}
          />
          <Route
            path='home'
            element={<Home />}
          />
          <Route
            path='task-list'
            element={<TaskList />}
          />
          <Route
            path='add-task'
            element={<AddTask />}
          />
        </Routes>
      </div>
    )
}
```

- navigation using react router
    - moving from one component to another
    - it does not reload the website
    - types
        - static navigation
            - the destination will never change
            - there is not need to add any code which needs to be executed on an event
            - achieved by using
            - e.g. register
        - dynamic
            - the navigation will be completed under certain conditions
            - requires a piece of code to execute to navigate to another component
            - e.g.

```
function Register() {
  // get the navigate function reference
  const navigate = useNavigate()
```

```
    const onRegister = () => {
      // ....
      // navigation to login screen
      navigate('/login')
    }
  }
```

- nested routing
    - used to load a component inside another component
    - e.g.

```
<Routes>
  <Route
    path='home'
    element={<Home />}
  >
    <Route
      path='task-list'
      element={<TaskList />}
    />
    <Route
      path='add-task'
      element={<AddTask />}
    />
  </Route>
</Routes>
```

        - to load the task-list, we have to use /home/task-list
        - to load the add-task, we have to use /home/add-task

vscode extensions

- https://marketplace.visualstudio.com/items/?itemName=NuclleaR.vscode-extension-auto-import
- https://marketplace.visualstudio.com/items/?itemName=formulahendry.auto-rename-tag
- https://marketplace.visualstudio.com/items/?itemName=streetsidesoftware.code-spell-checker
- https://marketplace.visualstudio.com/items/?itemName=rodrigovallades.es7-react-js-snippets
- https://marketplace.visualstudio.com/items/?itemName=sidthesloth.html5-boilerplate
-