Frontend React Notes

Commands, Examples, Project Structure & Base Application

Contents

What is React?

React is a JavaScript library for building user interfaces with reusable components.

React = UI = f(state). When state changes → UI updates automatically.

Commands for React Development

Create React project using Vite

```
npm create vite@latest my-react-app -- --template react
cd my-react-app
npm install
npm run dev
```

Common npm Commands

```
npm install
npm run dev
npm run build
npm run preview
```

React Project Structure

```
src/
  main.jsx              <-- mounts React into <div id="root">
  App.jsx               <-- main application logic

  components/           <-- optional reusable components
    Header.jsx
    TodoList.jsx

  App.css               <-- component-specific styles
  index.css             <-- global baseline styles
```

React projects follow a simple pattern:

- `main.jsx` starts the app.
- `App.jsx` contains the user interface.
- `components/` stores reusable UI pieces.

Analysis of main.jsx

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App'

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
)
```

`main.jsx` loads the React application and injects it into:

`<div id="root"> ... </div>`

Everything drawn by React appears inside this container.

Example App.jsx

```
import { useState } from 'react'

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

  return (
    <div>
      <h1>Counter</h1>
      <p>Value: {count}</p>

      <button onClick={() => setCount(count + 1)}>+</button>
      <button onClick={() => setCount(0)}>Reset</button>
    </div>
  )
}

export default App
```

This basic example demonstrates the essential frontend tools:

- JSX syntax (HTML-like code inside JavaScript)
- `useState` to store changing values
- event handlers such as `onClick`

These will be used in any React frontend.

Components Example

Header.jsx

```
function Header({ title }) {
  return (
    <header style={{
      background: '#0059B3',
      color: 'white',
      padding: '1rem'
    }}>
      <h1>{title}</h1>
    </header>
  )
}

export default Header
```

A component accepts `props` (data passed into it). This allows reuse of the same UI element in multiple places.

Todo List Example

```
import { useState } from 'react'

function TodoApp() {
  const [todos, setTodos] = useState([])
  const [text, setText] = useState('')

  const addTodo = () => {
    if (!text.trim()) return
    setTodos([...todos, text])
    setText('')
  }

  return (
    <div>
      <input value={text} onChange={e => setText(e.target.value)} />
      <button onClick={addTodo}>Add</button>

      <ul>
        {todos.map((t, i) => (
          <li key={i}>{t}</li>
        ))}
      </ul>
    </div>
  )
}

export default TodoApp
```

This example introduces several important frontend concepts:

- arrays stored in state
- updating arrays using the spread operator (`...`)
- rendering lists with `.map()`

These are fundamental tools in any React-based interface.

Fetching Data from a Backend

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

function Users() {
  const [users, setUsers] = useState([])

  useEffect(() => {
    fetch('http://localhost:8080/api/users')
      .then(res => res.json())
      .then(data => setUsers(data))
  }, [])

  return (
    <ul>
      {users.map(u => (
        <li key={u.id}>{u.name}</li>
      ))}
    </ul>
  )
}

export default Users
```

`useEffect` runs code when the component first loads. It is typically used to:

- fetch data from an API
- initialize state
- perform side effects

Summary

Core React concepts for building a frontend:

- Components = independent UI blocks.
- JSX = HTML-like syntax merged with JavaScript logic.
- `useState` = remembers values between renders.
- `useEffect` = runs side effects (e.g. API calls).
- Lists rendered with `.map()`.
- Styling via CSS files.
- Project started and served with Vite.

These concepts form the basis of all modern React frontends.