React Learning Notes with Code Examples

# Chapter 1: Start Here

React is a JavaScript library for building user interfaces. It allows us to create reusable UI components that update efficiently when data changes. Every React app typically starts with an index.js file that renders the root component (App) into the DOM.

// index.js  
import React from 'react';  
import ReactDOM from 'react-dom/client';  
import './index.css';  
import App from './App';  
  
const root = ReactDOM.createRoot(document.getElementById('root'));  
root.render(  
 <React.StrictMode>  
 <App />  
 </React.StrictMode>  
);

# Chapter 2: App & JSX

JSX is a syntax extension that allows us to write HTML-like code inside JavaScript. The App.js file is the main entry point where components are combined to form the UI.

// App.js (snippet)  
return (  
 <div className="App">  
 <Header title="Groceries List" />  
 <AddItem  
 newItem={newItem}  
 setNewItem={setNewItem}  
 handleSubmit={handleSubmit}  
 />  
 <SearchItem search={search} setSearch={setSearch} />  
 <Content  
 items={items.filter(item =>   
 (item.item).toLowerCase().includes(search.toLowerCase())  
 )}  
 handleCheck={handleCheck}  
 handleDelete={handleDelete}  
 />  
 <Footer length={items.length} />  
 </div>  
);

# Chapter 3: Functional Components

Functional components are simple JavaScript functions that return JSX. They receive data via props and can manage state with hooks.

// Header.js  
const Header = ({ title }) => {  
 return (  
 <header>  
 <h1>{title}</h1>  
 </header>  
 );  
};  
  
export default Header;

# Chapter 4: Applying CSS Styles

CSS styles can be applied by importing CSS files directly into React components. Styles are scoped globally unless CSS modules are used.

/\* App.css \*/  
.App {  
 display: flex;  
 flex-direction: column;  
 height: 100vh;  
 justify-content: space-between;  
}

# Chapter 5: Click Events

In React, click events are handled by attaching functions to onClick. These functions can update state or trigger other logic.

// LineItem.js  
<input  
 type="checkbox"  
 onChange={() => handleCheck(item.id)}  
 checked={item.checked}  
/>  
<label  
 style={(item.checked) ? { textDecoration: 'line-through' } : null}  
 onDoubleClick={() => handleCheck(item.id)}  
>  
 {item.item}  
</label>  
<button onClick={() => handleDelete(item.id)}>Delete</button>

# Chapter 6: useState Hook

The useState hook lets us add state to functional components. It returns a value and a function to update it.

// App.js  
import { useState } from 'react';  
  
const [items, setItems] = useState([]);  
const [newItem, setNewItem] = useState('');  
const [search, setSearch] = useState('');

# Chapter 7: Lists & Keys

React requires unique keys when rendering lists so it can efficiently update the UI when items change. while list is rendering key id is needed in react.

// ItemList.js  
const ItemList = ({ items, handleCheck, handleDelete }) => {  
 return (  
 <ul>  
 {items.map(item => (  
 <LineItem  
 key={item.id}  
 item={item}  
 handleCheck={handleCheck}  
 handleDelete={handleDelete}  
 />  
 ))}  
 </ul>  
 );  
};

# Chapter 8: Props & Prop Drilling

Props allow data to be passed from parent to child components. Prop drilling occurs when props are passed through multiple layers.

// Passing props from App.js  
<Content  
 items={items}  
 handleCheck={handleCheck}  
 handleDelete={handleDelete}  
/>  
  
// Receiving props in Content.js  
const Content = ({ items, handleCheck, handleDelete }) => {  
 return (  
 <>  
 {items.length ? (  
 <ItemList items={items} handleCheck={handleCheck} handleDelete={handleDelete} />  
 ) : (  
 <p style={{ marginTop: '2rem' }}>Your list is empty.</p>  
 )}  
 </>  
 );  
};

# Chapter 9: Controlled Component Inputs

Controlled inputs in React tie the input’s value to component state. Any change in the input updates the state via onChange.

// SearchItem.js  
<input  
 type="text"  
 placeholder="Search Items"  
 value={search}  
 onChange={(e) => setSearch(e.target.value)}  
/>

# Chapter 10: Project Challenge

By combining all learned concepts, the project implements a grocery list app with add, search, check, and delete features, demonstrating component structure, props, state, and events.

# Chapter 11: useEffect Hook

The useEffect hook runs side effects such as fetching data, updating the DOM, or saving to local storage. It can run on every render, on mount, or when dependencies change.

i) useEffect(() => {}) - call on every render

ii) useEffect(() => {}, []) - call on load render

iii) useEffect(() => {}, [items]) - call on item state change

// App.js  
import { useEffect } from 'react';  
  
useEffect(() => {  
 const fetchItems = async () => {  
 const response = await fetch(API\_URL);  
 const listItems = await response.json();  
 setItems(listItems);  
 };  
  
 fetchItems();  
}, []);

# Chapter 12: JSON Server

JSON Server is used to simulate a backend API. It stores data in a db.json file and exposes REST endpoints.

// db.json  
{  
 "items": [  
 { "id": 1, "checked": false, "item": "Milk" },  
 { "id": 2, "checked": false, "item": "Bread" }  
 ]  
}

# Chapter 13: Fetch API Data

React fetches data using the Fetch API or libraries like Axios. Data from JSON Server is retrieved and stored in state.

// apiRequest.js  
const apiRequest = async (url = '', optionsObj = null, errMsg = null) => {  
 try {  
 const response = await fetch(url, optionsObj);  
 if (!response.ok) throw Error('Please reload the app');  
 } catch (err) {  
 errMsg = err.message;  
 } finally {  
 return errMsg;  
 }  
};  
  
export default apiRequest;

# Chapter 14: CRUD Operations

React integrates with APIs to perform Create, Read, Update, Delete (CRUD) operations. Each operation updates both the UI state and the server data.

// App.js (Adding Item Example)  
const addItem = async (item) => {  
 const id = items.length ? items[items.length - 1].id + 1 : 1;  
 const myNewItem = { id, checked: false, item };  
 const listItems = [...items, myNewItem];  
 setItems(listItems);  
  
 const postOptions = {  
 method: 'POST',  
 headers: { 'Content-Type': 'application/json' },  
 body: JSON.stringify(myNewItem)  
 };  
  
 const result = await apiRequest(API\_URL, postOptions);  
 if (result) setFetchError(result);  
};