React Basics – Step-by-Step Guide

This document explains React basics, step by step.

---

1. Creating a React App

Step 1: Create the app

```
npx create-react-app@latest react-form-basic
```

What this does:

- Downloads a ready-made React setup
- Configures Babel, Webpack, ESLint
- Saves you from complex configuration

Think of it as:

> "Give me a working React project instantly."

Step 2: Go inside the project

```
cd react-form-basic
```

You must be inside the project folder to run React commands.

Step 3: Start the app

```
npm start
```

What happens:

- Starts a development server
- Opens browser at `http://localhost:3000`
- Auto-reloads when you change code

---

2. What is Data Binding?

Data binding means:

Connecting data (state) with UI (input fields)

Example:

```
const [name, setName] = useState("");

<input value={name} />
```

What's happening:

- `name` holds data
- Input shows the same data
- UI and data are always in sync

This is called **two-way data binding**.

---

3. What is Event Binding?

Event binding means:

Connecting user actions to functions

Example:

```
<input onChange={handleNameChange} />
```

Explanation:

- User types something
- `onChange` event fires
- `handleNameChange` function runs

Another example:

```
<button onClick={handleSubmit}>Login</button>
```

When the user clicks the button → function executes

---

4. Form Logic (How Form Works Internally)

Step-by-step logic

1. User types values
2. Values are stored in state
3. User clicks **Login**

4. `handleSubmit` runs
5. Validation happens
6. If valid → data stored
7. UI updates automatically

Example:

```
const handleSubmit = (e) => {
  e.preventDefault();
  setData([...data, formData]);
};
```

---

5. Form Validation (Simple Logic)

Validation means:

> Checking user input before saving

Example: Name validation

```
if (name.trim() === "") {
  setNameError("Name is required");
}
```

Example: Unique email check

```
data.some(item => item.email === email);
```

Meaning:

- Checks if email already exists
- Prevents duplicate entries

---

6. Displaying Data in a Table

Why table?

- Clean display
- Structured rows & columns

How React displays data

```
{data.map((item, index) => (
  <tr key={index}>
```

```
    <td>{item.name}</td>
    <td>{item.email}</td>
    <td>{item.gender}</td>
  </tr>
))}
```

Explanation:

- `map()` loops over array
- Each item becomes a table row
- React re-renders when data changes

---

7. Why Data Appears Automatically?

Because React is **state-driven**.

When you call:

```
setData(newData);
```

React automatically:

1. Updates state
2. Re-renders UI
3. Shows updated table

No manual DOM updates needed.

---

8. Key Concepts Summary

Concept	Meaning
React App	UI built with components
State	Data memory of component
Data Binding	State ↔ UI connection
Event Binding	User action → function
Validation	Input checks
map()	Display list data
JSX	HTML inside JavaScript

---

9. useEffect Hook (Very Important)

What is `useEffect`?

In simple words:

> `useEffect` is used to run code **after the component renders**.

A senior developer thinks of `useEffect` as:

> "Do side work like API calls, subscriptions, or logging when UI is ready."

When do we use `useEffect`?

- Fetch data from API
- Call backend services
- Read from localStorage
- Run code on page load
- React to state changes

Basic Syntax

```
useEffect(() => {
  // code to run
}, []);
```

Explanation:

- First argument → function to run
- Second argument → dependency array

Dependency Array Explained

Dependency	Meaning
`[]`	Run once (on page load)
`[state]`	Run when state changes
No array	Run on every render (not recommended)

---

10. useEffect Example – Fetch API Data and Display in Table

We will:

1. Fetch data from an API
2. Store it in state
3. Display it in a table

Step 1: Create State

```
const [users, setUsers] = useState([]);
```

This state will store API data.

Step 2: useEffect with Fetch API

```
useEffect(() => {
  fetch("https://jsonplaceholder.typicode.com/users")
    .then((response) => response.json())
    .then((data) => {
      setUsers(data);
    })
    .catch((error) => {
      console.error("Error fetching data", error);
    });
}, []);
```

How This Works (Simple Flow)

1. Component loads
2. `useEffect` runs once
3. API is called
4. Data is received
5. `setUsers` updates state
6. Table updates automatically

Step 3: Display API Data in Table

```
<table border="1" cellPadding="8">
  <thead>
    <tr>
      <th>ID</th>
      <th>Name</th>
```

```
      <th>Email</th>
    </tr>
  </thead>
  <tbody>
    {users.map((user) => (
      <tr key={user.id}>
        <td>{user.id}</td>
        <td>{user.name}</td>
        <td>{user.email}</td>
      </tr>
    ))}
  </tbody>
</table>
```

---

11. Complete Example (useEffect + Fetch + Table)

```
import React, { useEffect, useState } from "react";

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

  useEffect(() => {
    fetch("https://jsonplaceholder.typicode.com/users")
      .then((res) => res.json())
      .then((data) => setUsers(data));
  }, []);

  return (
    <div>
      <h3>User List</h3>
      <table border="1" cellPadding="8">
        <thead>
          <tr>
            <th>Name</th>
            <th>Email</th>
          </tr>
        </thead>
        <tbody>
          {users.map((u) => (
            <tr key={u.id}>
              <td>{u.name}</td>
              <td>{u.email}</td>
            </tr>
          ))}
        </tbody>
      </table>
    </div>
  );
```

```
}

export default UsersTable;
```

---

12. Senior Developer Tips for useEffect

- Always use dependency array
- Never update state blindly inside effect
- One effect = one responsibility
- Avoid API calls in render

---

13. Final Thought

If you understand:

- `useState`
- `useEffect`
- `onChange`
- `onSubmit`
- `map()`

👉You understand **most real-world React development**.

React is about **state + effects + UI**.

Keep practicing. You're doing great 😥