React for Flutter Developers

A quick introduction to React fundamentals

What is React?

- JavaScript library for building UIs
- **Component-based** architecture (like Flutter widgets)
- **Declarative** UI (similar to Flutter)
- Uses **Virtual DOM** for efficient updates

Flutter equivalent: Widget tree

JSX: JavaScript + HTML

```
// JSX allows HTML-like syntax in JavaScript
const greeting = <h1>Hello, World!</h1>;

// With JS expressions
const name = "Alice";
const greeting = <h1>Hello, {name}!</h1>;
```

Flutter equivalent:

```
Text('Hello, $name!')
```

Components: The Building Blocks

Functional Components (modern approach)

```
function Greeting(props) {
  return <h1>Hello, {props.name}!</h1>;
}

// Arrow function syntax
const Greeting = (props) => {
  return <h1>Hello, {props.name}!</h1>;
};
```

Flutter equivalent: StatelessWidget

Props: Passing Data Down

```
function UserCard(props) {
  return (
    <div>
      <h2>{props.name}</h2>
      <p>Age: {props.age}</p>
    </div>
  );
}

// Usage
// We should use JS expression {25} in JSX
<UserCard name="Alice" age={25} />
```

Flutter equivalent: Constructor parameters in widgets

Props Destructuring

```
// Cleaner syntax
function UserCard({ name, age }) {
  return (
    <div>
      <h2>{name}</h2>
      <p>Age: {age}</p>
    </div>
  );
}

// Usage
<UserCard name="Alice" age={25} />
```

More concise and readable!

State: Managing Component Data

```
import { useState } from 'react';

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

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>
        Increment
      </button>
    </div>
  );
}
```

Flutter equivalent: StatefulWidget with setState()

useState Hook

- `useState` returns an array: `[value, setter]`
- Initial value passed as argument
- Setter function triggers re-render

```
const [count, setCount] = useState(0);
//      ^         ^                ^
//    value    setter        initial value
```

Key difference: Unlike Flutter's setState, you call the setter directly

Multiple State Variables

```
function LoginForm() {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [isLoading, setIsLoading] = useState(false);

  return (
    <form>
      <input
        value={email}
        onChange={(e) => setEmail(e.target.value)}
      />
      <input
        value={password}
        onChange={(e) => setPassword(e.target.value)}
      />
    </form>
  );
}
```

Event Handling

```
function Button() {
  const handleClick = () => {
    console.log('Button clicked!');
  };

  return <button onClick={handleClick}>Click me</button>;
}

// Inline handler
<button onClick={() => console.log('Clicked!')}>
  Click me
</button>
```

Flutter equivalent: onPressed, onTap callbacks

useEffect Hook: Side Effects

```
import { useEffect } from 'react';

function DataFetcher() {
  const [data, setData] = useState(null);

  useEffect(() => {
    // This runs after component mounts
    fetchData().then(result => setData(result));
  }, []); // Empty array = run once on mount

  return <div>{data}</div>;
}
```

Flutter equivalent: initState(), didUpdateWidget()

```
function MyComponent() {
  useEffect(() => {
    // Component mounted (1)
    return () => {
      // Component will unmount (2)
    };
  }, []);
```

The return () => { ... } is a cleanup function in React: it runs before component unmounts

Flutter equivalent: dispose()

useEffect Dependencies

```
useEffect(() => {
  console.log('Runs on every render');
});

useEffect(() => {
  console.log('Runs once on mount');
}, []); // Empty dependencies
```

Run this effect only when the variable count changes.

```
useEffect(() => {
  console.log('Runs when count changes');
}, [count]); // Specific dependencies
```

Conditional Rendering

```
function Greeting({ isLoggedIn }) {
  return (
    <div>
      {isLoggedIn ? (
        <h1>Welcome back!</h1>
      ) : (
        <h1>Please sign in</h1>
      )}
    </div>
  );
}

// Or with &&
{isLoggedIn && <h1>Welcome back!</h1>}
```

Flutter equivalent: Conditional expressions in build()

Lists and Keys

```
function UserList({ users }) {
  return (
    <ul>
      {users.map(user => (
        <li key={user.id}>
          {user.name}
        </li>
      ))}
    </ul>
  );
}
```

Flutter equivalent: ListView.builder with key parameter

Important: Always provide a unique `key` for list items

```dart
import 'package:flutter/material.dart';

class UserList extends StatelessWidget {
  final List<Map<String, dynamic>> users;

  const UserList({super.key, required this.users});

  @override
  Widget build(BuildContext context) {
    return ListView(
      children: users.map((user) {
        return ListTile(
          key: ValueKey(user['id']),
          title: Text(user['name']),
        );
      }).toList(),
    );
  }
}
```

Component Lifecycle (with Hooks)

```
useEffect(() => {
  // Mount: Component appears
  console.log('Component mounted');

  return () => {
    // Cleanup: Component disappears
    console.log('Component will unmount');
  };
}, []);
```

Flutter equivalent: initState() and dispose()

React vs Flutter: Quick Comparison

React	Flutter
JSX	Dart widget tree
Props	Constructor parameters
useState	setState()
useEffect	initState, didUpdateWidget
Functional components	StatelessWidget
Components with state	StatefulWidget

Example: Todo App Structure

```
function TodoApp() {
  const [todos, setTodos] = useState([]);
  const [input, setInput] = useState('');

  const addTodo = () => {
    setTodos([...todos, { id: Date.now(), text: input }]);
    setInput('');
  };

  return (
    <div>
      <input value={input} onChange={e => setInput(e.target.value)} />
      <button onClick={addTodo}>Add</button>
      <TodoList todos={todos} />
    </div>
  );
}
```

Key Takeaways

1. **Components** are like Flutter widgets
2. **Props** flow down (immutable)
3. **State** is local and mutable
4. **Hooks** (useState, useEffect) manage state and side effects
5. **JSX** is declarative like Flutter's UI
6. **Re-rendering** happens automatically when state changes

Resources

- React Docs: https://react.dev
- React Tutorial: https://react.dev/learn
- Thinking in React: https://react.dev/learn/thinking-in-react

Next: React Native for mobile development