React Native - CounterApp

Filter by Topic

Structure

Importance of useState()

Code Snippet:

Copy Code

```
// Step-by-Step Instructions for a JavaScript Expo Application
// Create the Expo Project: Run the following command to create a new Expo project
npx create-expo-app@latest CounterApp


// Navigate to the Project Directory:
cd CounterApp

// Reset The Project
npm run reset-project

// Run
npx expo start

// Project Structure: The created project will have a structure similar to this:

CounterApp/
├── app/
│   └── index.jsx
├── assets/
├── node_modules/
├── package.json
├── app.json
├── babel.config.js
└── yarn.lock (or package-lock.json if using npm)
```

```
// Creating the styles.js File: Create a new file named styles.js in the app direc
```

index.jsx

app/index.jsx

Code Snippet:

Copy Code

```
// app/index.jsx
import React, { useEffect, useState } from 'react';
import { View, Text, Button } from 'react-native';
import styles from './styles';

const CounterWithoutState = () => {
    let count = 0;

    const handlePress = () => {
        count += 1;
        console.log("Count without useState:", count);
    };

    const handlePressDecrement = () => {
        count -= 1;
        console.log("Count without useState:", count);
    };

    const handlePressReset = () => {
      count = 0;
      console.log("Count without useState:", count);
  };

    return (
        <View style={styles.counterContainer}>
            <Text style={styles.counterText}>Count without useState: {count}</Text
            <View style={styles.buttonContainer}>
              <View style={styles.buttonWrapper}>
                  <Button title="Increment" onPress={handlePress} />
              </View>
              <View style={styles.buttonWrapper}>
```

```
                    <Button title="Decrement" onPress={handlePressDecrement} />
                </View>
                <View style={styles.buttonWrapper}>
                    <Button title="Reset" onPress={handlePressReset} />
                </View>


            </View>
        </View>
    );
};

const CounterWithState = () => {
    const [count, setCount] = useState(0);

    const handlePress = () => {
        setCount(count + 1);
        console.log("Count without useState:", count);
    };

    const handlePressDecrement = () => {
        setCount(count - 1);
        console.log("Count without useState:", count);
    };

    const handlePressReset = () => {
        setCount(0);
        console.log("Count without useState:", count);
    };

    // useEffect(() => {
    //     console.log("Count updated:", count);
    // }, [count]);

    return (
      <View style={styles.counterContainer}>
          <Text style={styles.counterText}>Count with useState: {count}</Text>
          <View style={styles.buttonContainer}>
              <View style={styles.buttonWrapper}>
                  <Button title="Increment" onPress={handlePress} />
              </View>
              <View style={styles.buttonWrapper}>
                  <Button title="Decrement" onPress={handlePressDecrement} />
              </View>
              <View style={styles.buttonWrapper}>
                  <Button title="Reset" onPress={handlePressReset} />
```

```
                </View>
            </View>
        </View>
    );
};

const App = () => {
    return (
        <View style={styles.container}>
            <Text style={styles.title}>Counter App</Text>
            <CounterWithoutState />
            <CounterWithState />
        </View>
    );
};

export default App;
```

styles.js

app/styles.js

Code Snippet:

Copy Code

```
// app/styles.js
import { StyleSheet } from 'react-native';

const styles = StyleSheet.create({
    container: {
        flex: 1,
        justifyContent: 'center',
        alignItems: 'center',
        backgroundColor: '#f5fcff',
        padding: 20,
    },
    title: {
        fontSize: 24,
        fontWeight: 'bold',
        marginBottom: 20,
    },
    counterContainer: {
        marginVertical: 10,
```

```
        alignItems: 'center',
    },
    counterText: {
        fontSize: 20,
        marginBottom: 10,
    },
    buttonContainer: {
        flexDirection: 'row',
        justifyContent: 'center', // Center buttons horizontally
    },
    buttonWrapper: {
        marginHorizontal: 10, // Adds horizontal margin between buttons
    },
});

export default styles;
```

package.json

package.json

Code Snippet:

Copy Code

```
{
  "name": "counterapp",
  "main": "expo-router/entry",
  "version": "1.0.0",
  "scripts": {
    "start": "expo start",
    "reset-project": "node ./scripts/reset-project.js",
    "android": "expo start --android",
    "ios": "expo start --ios",
    "web": "expo start --web",
    "test": "jest --watchAll",
    "lint": "expo lint"
  },
  "jest": {
    "preset": "jest-expo"
  },
  "dependencies": {
    "@expo/vector-icons": "^14.0.2",
    "@react-navigation/native": "^6.0.2",
```

```
    "expo": "~51.0.28",
    "expo-constants": "~16.0.2",
    "expo-font": "~12.0.9",
    "expo-linking": "~6.3.1",
    "expo-router": "~3.5.23",
    "expo-splash-screen": "~0.27.5",
    "expo-status-bar": "~1.12.1",
    "expo-system-ui": "~3.0.7",
    "expo-web-browser": "~13.0.3",
    "react": "18.2.0",
    "react-dom": "18.2.0",
    "react-native": "0.74.5",
    "react-native-gesture-handler": "~2.16.1",
    "react-native-reanimated": "~3.10.1",
    "react-native-safe-area-context": "4.10.5",
    "react-native-screens": "3.31.1",
    "react-native-web": "~0.19.10"
  },
  "devDependencies": {
    "@babel/core": "^7.20.0",
    "@types/jest": "^29.5.12",
    "@types/react": "~18.2.45",
    "@types/react-test-renderer": "^18.0.7",
    "jest": "^29.2.1",
    "jest-expo": "~51.0.3",
    "react-test-renderer": "18.2.0",
    "typescript": "~5.3.3"
  },
  "private": true
}
```

useState()

What is useState?

In React, useState is a Hook that allows us to add state to functional components. Before hooks, managing state was only possible in class components.

However, with hooks like useState, we can create, update, and manage state directly within functional components.

Key points about useState:

1. It allows functional components to have their own state, similar to this.state in class components.
2. useState is initialized with a default value, and it returns an array containing:

- The current state value.
- A function to update that state value.

Basic Syntax of useState

const [state, setState] = useState(initialState);

- **state**: The current state value.
- **setState**: A function to update the state.
- **initialState**: The initial value of the state.

Code Snippet:

Copy Code

```
// React JS (Web App)
// Example: Creating a Simple Counter with useState
// Let's create a simple counter that:

// Starts with an initial count of 0.
// Has a button to increment the count by 1.

import React, { useState } from 'react';

function Counter() {
  // Declare a state variable "count" with initial value of 0
  const [count, setCount] = useState(0);

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

```
export default Counter;


// Explanation of Code
import React, { useState } from 'react';

// We import React along with the useState hook from the React library. The { useS
const [count, setCount] = useState(0);

// We declare a new state variable named count with an initial value of 0.
// useState(0) is called with 0 as its initial value, meaning that count will star
// setCount is the function that updates the value of count.
// Every time setCount is called with a new value, React re-renders the component
<p>Current Count: {count}</p>

// Displays the current value of count in the browser.
<button onClick={() => setCount(count + 1)}>Increment</button>

// A button with an onClick event handler that increments count by 1 whenever it’s
// The function setCount(count + 1) is called to update count to its current value




// Example Output (React.js)
// When the component is rendered, you see:

// The current count displayed.
// The "Increment" button that increases the count each time it's clicked.




// Native App
// React Native Code Example
// In React Native, the code is almost identical, but we need to use React Native

import React, { useState } from 'react';
import { View, Text, Button } from 'react-native';

function Counter() {
  // Declare a state variable "count" with initial value of 0
  const [count, setCount] = useState(0);
```

```
  return (
    <View style={{ alignItems: 'center', marginTop: 20 }}>
      <Text style={{ fontSize: 20 }}>Current Count: {count}</Text>
      <Button title="Increment" onPress={() => setCount(count + 1)} />
    </View>
  );
}

export default Counter;


// Explanation of Code
import { View, Text, Button } from 'react-native';

// We use View, Text, and Button components for layout and text display, as they a
<View style={{ alignItems: 'center', marginTop: 20 }}>

// Adds some basic styling to center-align content and add space at the top.
<Text style={{ fontSize: 20 }}>Current Count: {count}</Text>

// Displays the current count with a larger font size using inline styles.
<Button title="Increment" onPress={() => setCount(count + 1)} />

// Button component in React Native uses a title prop for the button text, and onP
// The onPress handler calls setCount(count + 1) to increase the count by 1.


// Additional Notes on useState
// Updating State Asynchronously: State updates in React are asynchronous, meaning
// This is an important behavior to remember as you work with dynamic data or chai



// Multiple State Variables:
// You can use useState multiple times within a component to manage multiple indep

const [name, setName] = useState('');
const [age, setAge] = useState(0);
```

Managing Complex State with useState in React and React Native

Using useState with Objects

When you use useState with objects, you need to remember that React's state updates are **shallow merges**.

This means that if you want to update only a part of the object, you need to spread the existing object's properties into the new state value to avoid accidentally overwriting other properties.

Code Snippet:

Copy Code

```
// Example: Creating a User Profile with Multiple Properties
// We'll create a user profile form with two fields: name and age.

// React.js Code Web App
import React, { useState } from 'react';

function UserProfile() {
  // Initialize state as an object with two properties
  const [user, setUser] = useState({ name: '', age: '' });

  // Function to handle changes in the input fields
  const handleChange = (e) => {
    const { name, value } = e.target;
    setUser((prevUser) => ({
      ...prevUser,      // Copy previous state
      [name]: value,    // Update the field that triggered the change
    }));
  };

  return (
    <div>
      <h2>User Profile</h2>
      <label>
        Name:
        <input type="text" name="name" value={user.name} onChange={handleChange} />
      </label>
      <br />
      <label>
        Age:
        <input type="number" name="age" value={user.age} onChange={handleChange} />
      </label>
      <br />
      <p>User Name: {user.name}</p>
```

```
      <p>User Age: {user.age}</p>
    </div>
  );
}

export default UserProfile;

// Explanation of Code
const [user, setUser] = useState({ name: '', age: '' });
// We initialize user as an object with name and age properties, both set to an em
const handleChange = (e) => {...}
// handleChange takes the event object e, extracts name and value from the input f
setUser((prevUser) => ({ ...prevUser, [name]: value }));

// Here, setUser updates the user state without overwriting the entire object by u
// The [name]: value syntax dynamically sets the object property.




// React Native Code Example
// The same logic applies to React Native, but with TextInput components instead o
import React, { useState } from 'react';
import { View, Text, TextInput } from 'react-native';

function UserProfile() {
  const [user, setUser] = useState({ name: '', age: '' });

  const handleChange = (key, value) => {
    setUser((prevUser) => ({
      ...prevUser,
      [key]: value,
    }));
  };

  return (
    <View style={{ padding: 20 }}>
      <Text>User Profile</Text>
      <TextInput
        placeholder="Name"
        value={user.name}
        onChangeText={(value) => handleChange('name', value)}
        style={{ borderBottomWidth: 1, marginBottom: 10 }}
      />
      <TextInput
```

```
          placeholder="Age"
          keyboardType="numeric"
          value={user.age}
          onChangeText={(value) => handleChange('age', value)}
          style={{ borderBottomWidth: 1, marginBottom: 10 }}
        />
        <Text>User Name: {user.name}</Text>
        <Text>User Age: {user.age}</Text>
      </View>
    );
}

export default UserProfile;

// Explanation of Code
<TextInput ... />
// In React Native, we use TextInput with an onChangeText prop, which directly pro
// The function handleChange is called with the key (either name or age) and the u


// Using useState with Arrays
// useState can also manage arrays, making it useful for lists or collections of i
// Let’s create a simple to-do list where users can add new tasks and display them
```

To Do List Example

React JS Web & Native

Code Snippet:

Copy Code

```
// Example: To-Do List with Array State
// React.js Code Web App
import React, { useState } from 'react';

function TodoList() {
  const [tasks, setTasks] = useState([]);
  const [task, setTask] = useState('');

  const addTask = () => {
```

```
    setTasks((prevTasks) => [...prevTasks, task]);
    setTask(''); // Clear input after adding
  };

  return (
    <div>
      <h2>To-Do List</h2>
      <input
        type="text"
        value={task}
        onChange={(e) => setTask(e.target.value)}
        placeholder="Enter a task"
      />
      <button onClick={addTask}>Add Task</button>
      <ul>
        {tasks.map((item, index) => (
          <li key={index}>{item}</li>
        ))}
      </ul>
    </div>
  );
}

export default TodoList;

// Explanation of Code
const [tasks, setTasks] = useState([]);

// tasks starts as an empty array to hold the list of tasks.
const addTask = () => {...}

// When addTask is called, it adds the current task value to the tasks array.
// We use ...prevTasks to retain previous tasks, then append the new task.




// React Native Code Example
// The code is similar but uses TextInput and FlatList to display tasks.
import React, { useState } from 'react';
import { View, Text, TextInput, Button, FlatList } from 'react-native';

function TodoList() {
  const [tasks, setTasks] = useState([]);
  const [task, setTask] = useState('');
```

```
  const addTask = () => {
    setTasks((prevTasks) => [...prevTasks, task]);
    setTask('');
  };

  return (
    <View style={{ padding: 20 }}>
      <Text>To-Do List</Text>
      <TextInput
        placeholder="Enter a task"
        value={task}
        onChangeText={setTask}
        style={{ borderBottomWidth: 1, marginBottom: 10 }}
      />
      <Button title="Add Task" onPress={addTask} />
      <FlatList
        data={tasks}
        keyExtractor={(item, index) => index.toString()}
        renderItem={({ item }) => <Text>{item}</Text>}
      />
    </View>
  );
}

export default TodoList;

// Explanation of Code
// FlatList Used for displaying lists in React Native.
// keyExtractor uses the index as a unique key for each task.
// renderItem Describes how each item in tasks should be displayed.
```