React Native State, Props & Event Handling: Beginner to Advanced

Subtitle: Learn how to manage data and interactions in React Native apps effectively.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app

Introduction

State, props, and event handling are core concepts for building dynamic React Native apps. Understanding them lets you create interactive screens, manage data flow, and respond to user actions efficiently. This guide walks beginners through practical examples and exercises.

Step 1: Understanding State

State is **mutable data** that determines how a component behaves or renders.

- Use `useState` hook in functional components.
- Changing state **triggers a re-render** of the component.

Example:

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

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

  return (
    <View style={{ padding: 20 }}>
      <Text>Count: {count}</Text>
      <Button title="Increase" onPress={() => setCount(count + 1)} />
    </View>
  );
}
```

Exercise: Create a counter app with **Increment** and **Decrement** buttons.

Step 2: Understanding Props

Props are **read-only data passed from parent to child components**.

- Props allow **reusable components** with different values.
- Props cannot be modified by the child.

Example:

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

export default function App() {
  return <Greeting name="Hafsa" />;
}
```

Exercise: Build a **Card component** that accepts `title` and `description` as props and displays them.

Step 3: Event Handling

React Native uses **functions as event handlers**:

- Common events: `onPress`, `onChangeText`, `onScroll`.
- Pass **functions** to handle actions.

Example:

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

export default function App() {
  const [name, setName] = useState('');

  return (
    <View style={{ padding: 20 }}>
      <TextInput
        placeholder="Enter your name"
        value={name}
        onChangeText={text => setName(text)}
        style={{ borderWidth: 1, padding: 8, marginBottom: 10 }}
      />
      <Button title="Greet" onPress={() => alert(`Hello, ${name}!`)} />
```

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

Exercise: Create a **form** with two TextInput fields for **email** and **password** and a submit button that alerts the input values.

Step 4: Combining State, Props, and Events

- Use **state** for dynamic data.
- Use **props** to pass data to child components.
- Use **event handlers** to update state or trigger actions.

Example:

```
function Task({ task, onComplete }) {
  return (
    <Button title={task} onPress={() => onComplete(task)} />
  );
}

export default function App() {
  const [tasks, setTasks] = useState(['Buy milk', 'Walk dog']);

  const handleComplete = task => {
    setTasks(tasks.filter(t => t !== task));
  };

  return (
    <View>
      {tasks.map(task => (
        <Task key={task} task={task} onComplete={handleComplete} />
      ))}
    </View>
  );
}
```

Exercise: Build a **To-Do list app** where pressing a task removes it from the list.

Step 5: Best Practices

- Keep state minimal and lift it **up** when multiple components share data.
- Use **descriptive prop names** for clarity.
- Prefer **functional components with hooks** over class components.
- Always test events on **both iOS and Android**.

State, props, and events form the backbone of interactive React Native apps. Mastering these allows you to build **dynamic, user-friendly, and reusable components** efficiently.

Visit **haas.dev** for more React Native guides, tutorials, and step-by-step projects.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app
