Lecture 04

The Weather Application

Contents

- Creating the Weather App
- Learning some components
  - `ImageBackground`, `KeyboardAvoidingView`
  - `StatusBar`, `ActivityIndicator`, `TextInput`
- Consuming APIs
  - Weather API
  - Geocoding API

Handling Events in React Native

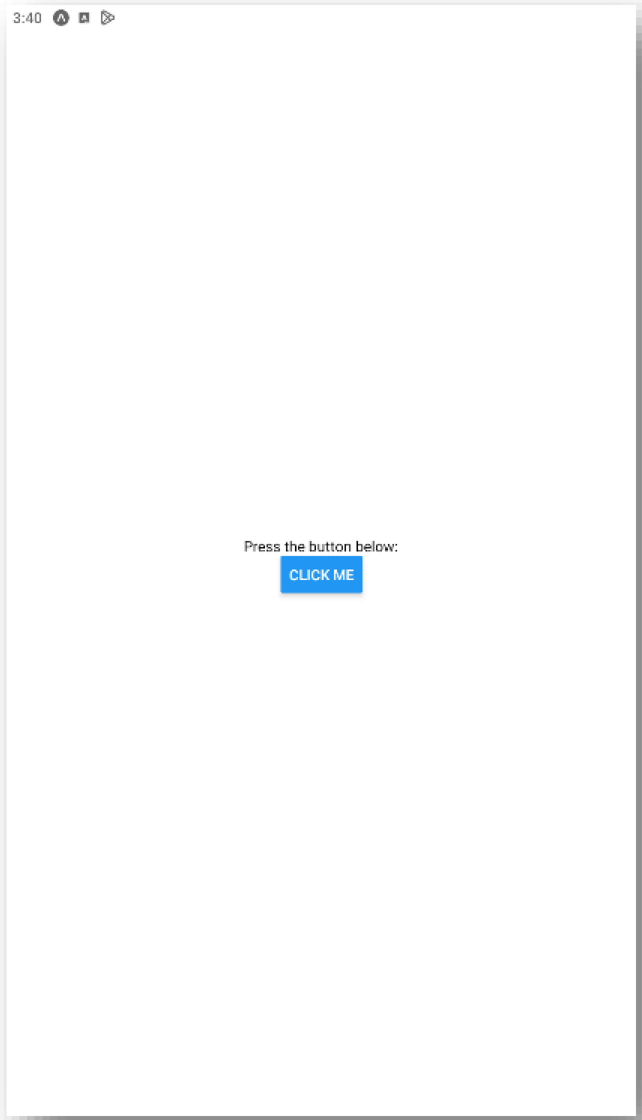
- In React Native, event handling works similarly to React for the web. Events such as user interactions (**onPress**, **onChangeText**, etc.) trigger state updates, which cause re-renders.

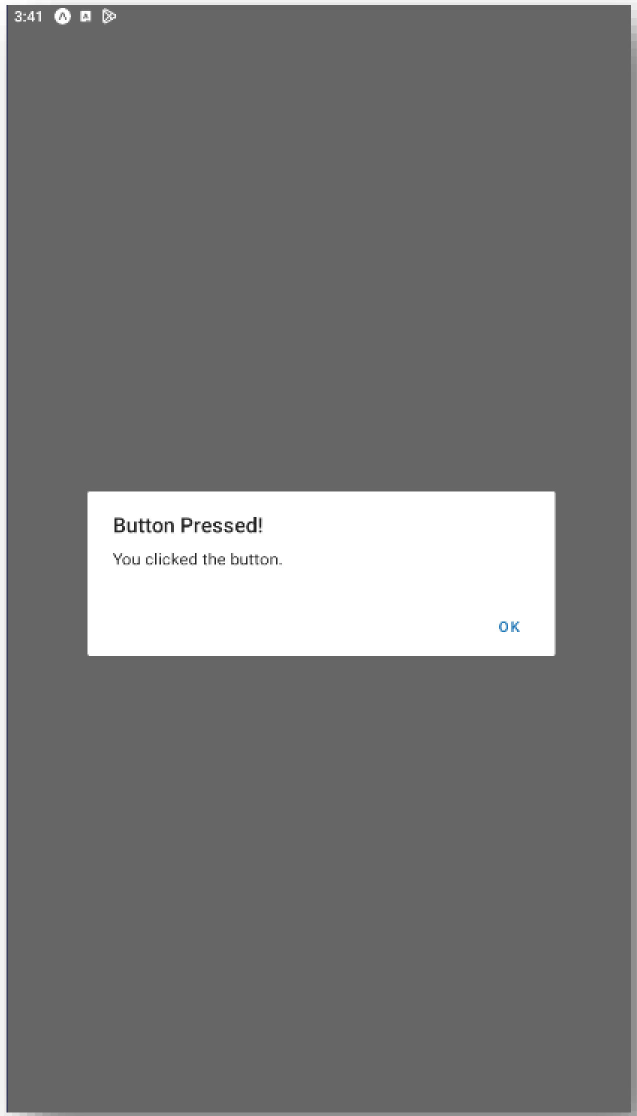
Events	Description
`onPress`	Triggered when a user taps a button or `Touchable` component.
`onChangeText`	Fires when text changes in an `TextInput` field.
`onSubmitEditing`	Executes when the user presses "Enter" or submits input.
`onLongPress`	Activated when a user presses and holds a button.
`onFocus / onBlur`	Used for handling focus state in input fields.

Handling Events in React Native

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

const HandleEventDemo = () => {
  const handlePress = () => {
    Alert.alert('Button Pressed!', 'You clicked the button.');
  };
  return (
    <View style={styles.container}>
      <Text>Press the button below:</Text>
      <Button title="Click Me" onPress={handlePress} />
    </View>
  );
};
export default HandleEventDemo;
```

Handling Events in React Native





Button Pressed!

You clicked the button.

Recall: The `useState` hook

- React Native uses **hooks** to manage state in functional components.
- The `useState` hook allows you to add state to a functional component.
- Syntax:

```
const [state, setState] = useState(initialValue);
```

Example: Managing State with `useState`

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

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

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

export default CounterApp;
```

- The `count` state stores a number.
- `setCount` updates the value when the button is clicked.

Syntax of `useEffect`

```
useEffect(() => {
    // Code to run on mount
    return () => {
      // Cleanup function (like componentWillUnmount)
    };
}, [dependencies]); // Dependencies control re-runs
```

Example: Using `useEffect` for Lifecycle Events

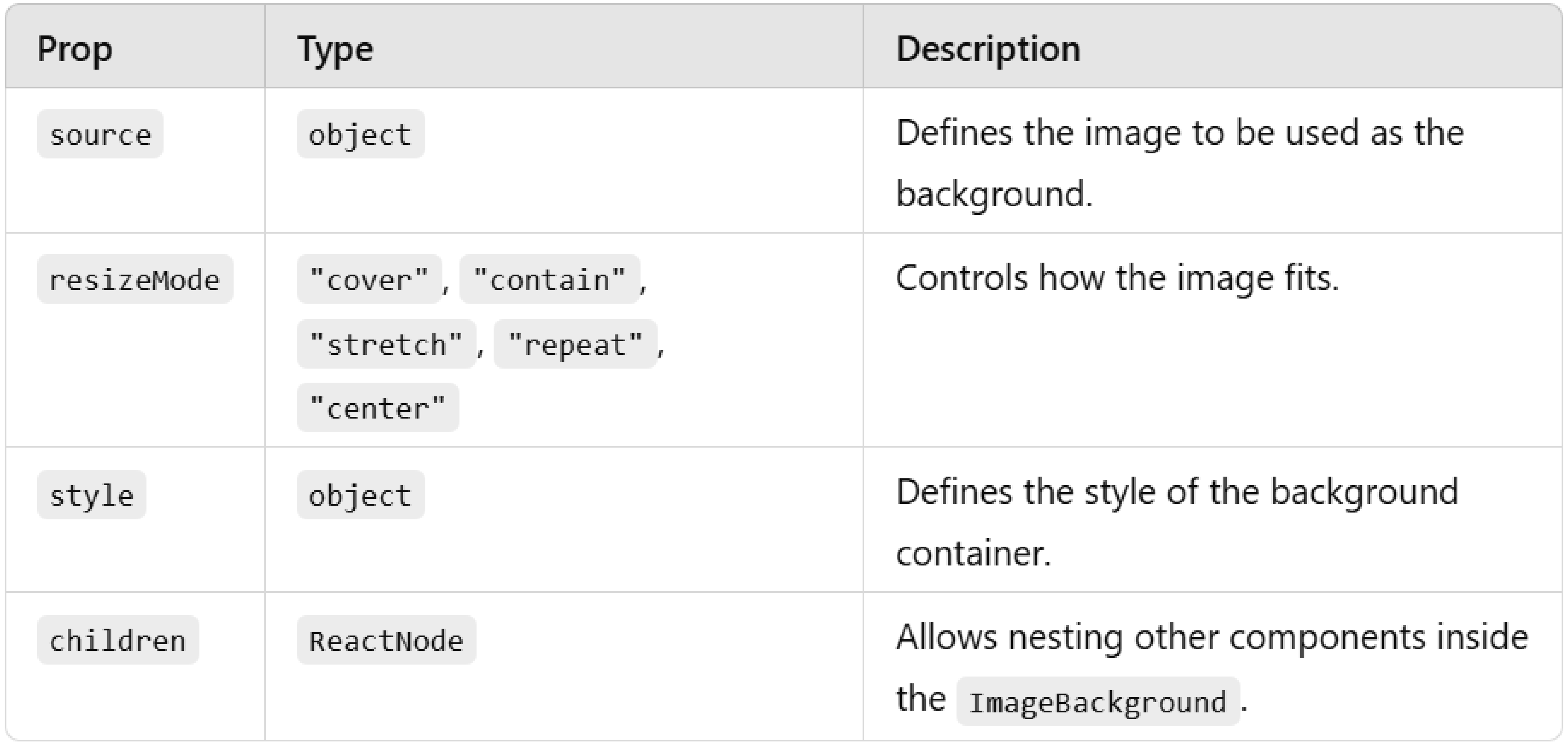
```
import React, { useState, useEffect } from 'react';
import { View, Text, Button } from 'react-native';

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

  useEffect(() => {
    console.log('Component Mounted');

    return () => {
      console.log('Component Unmounted');
    };
  }, []);
}
```

Core Component: `ImageBackground`

- Similar to the `background-image` CSS property on the Web.
- The `<ImageBackground>` component creates an image background
  - Has the same props as `<Image>`
  - Its children will be displayed on top of it
  - It doesn’t work well with border (shouldn’t give it border)
  - Should put it inside a View to set its size correctly

Core Component: `ImageBackground`

Some props:



Prop	Type	Description
`source`	`object`	Defines the image to be used as the background.
`resizeMode`	`"cover"`, `"contain"`, `"stretch"`, `"repeat"`, `"center"`	Controls how the image fits.
`style`	`object`	Defines the style of the background container.
`children`	`ReactNode`	Allows nesting other components inside the `ImageBackground`.

Example

```
import { ImageBackground, Text, StyleSheet } from 'react-native';

export default function App() {
    return (
        <ImageBackground
            source={require('./assets/background.jpg')}
            style={styles.background}
        >
            <Text style={styles.text}>Weather App</Text>
        </ImageBackground>
    );
}
```

- Use it as a wrapper to display content on top of an image.
- The `source` prop specifies the image.
- Use `resizeMode` to control how the image fits.

Core component: `StatusBar`

- `StatusBar` allows customization of the system's status bar, including color, visibility, and theme.
- Use `barStyle` to set text color (light-content, dark-content).
- Use `backgroundColor` to change the background color on Android.

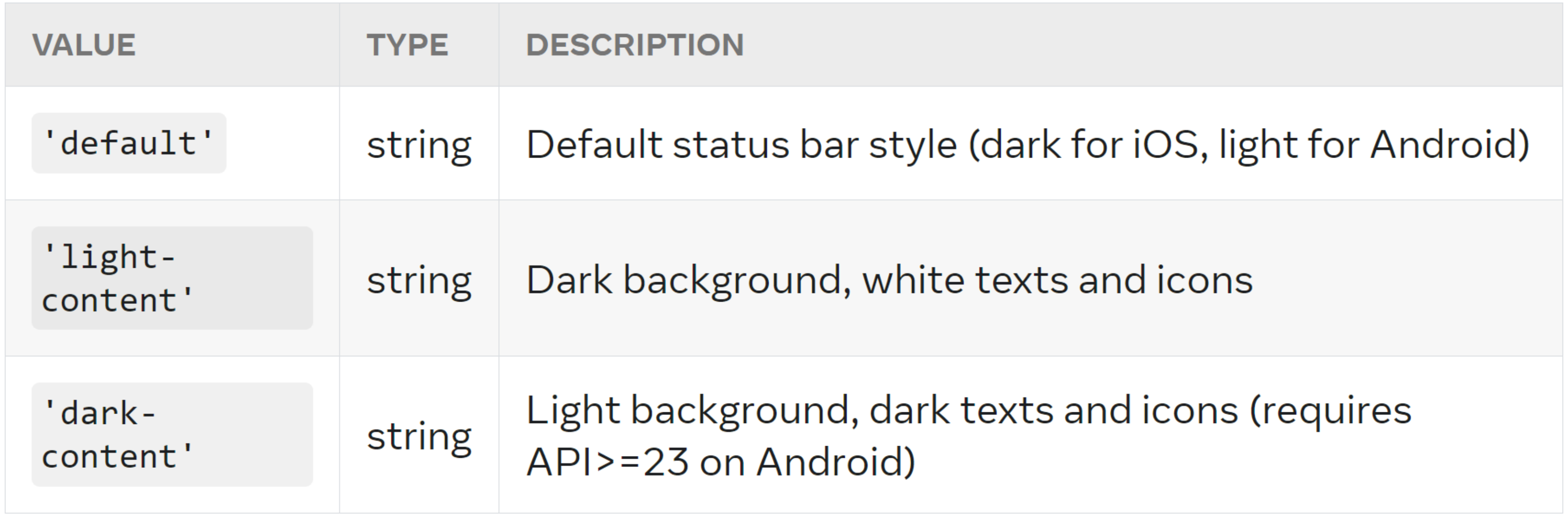
```
import { StatusBar } from 'react-native';

export default function App() {
    return (
        <>
            <StatusBar barStyle="light-content" backgroundColor="#000" />
        </>
    );
}
```

There is also a `StatusBar` component from `expo`, which is slightly different

Core component: `StatusBar`

- Some props:

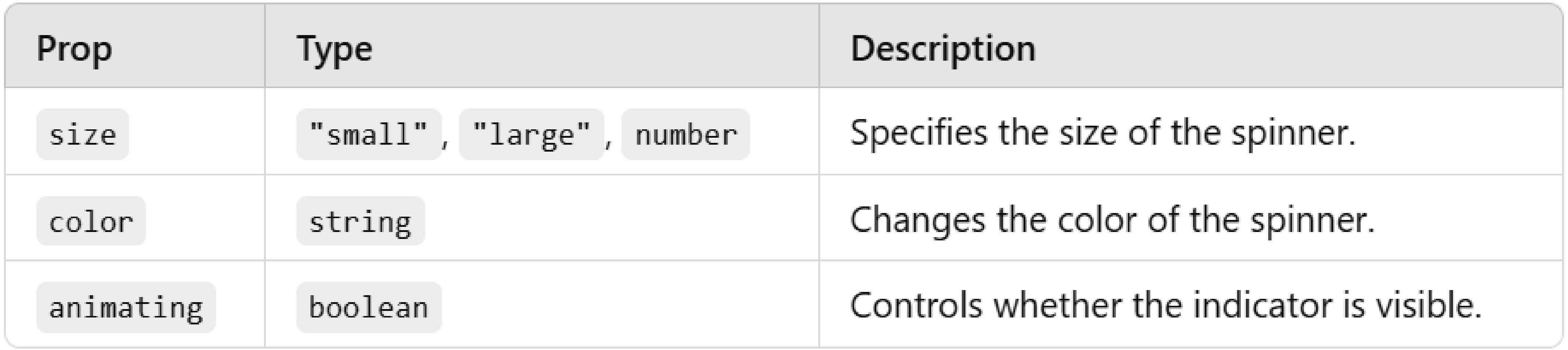
• `hidden` (default: `false`)	Controls the visibility of the status bar
• `barStyle`	3 styles to choose from (see table below)
• `animated`	Animated property changes (supports `barStyle`, `backgroundColor` & `hidden`)
• `backgroundColor`	Android only



VALUE	TYPE	DESCRIPTION
`'default'`	string	Default status bar style (dark for iOS, light for Android)
`'light-content'`	string	Dark background, white texts and icons
`'dark-content'`	string	Light background, dark texts and icons (requires API>=23 on Android)

Core component: `ActivityIndicator`

- `ActivityIndicator` is used to show a loading spinner when fetching data, like this:
- Some props:



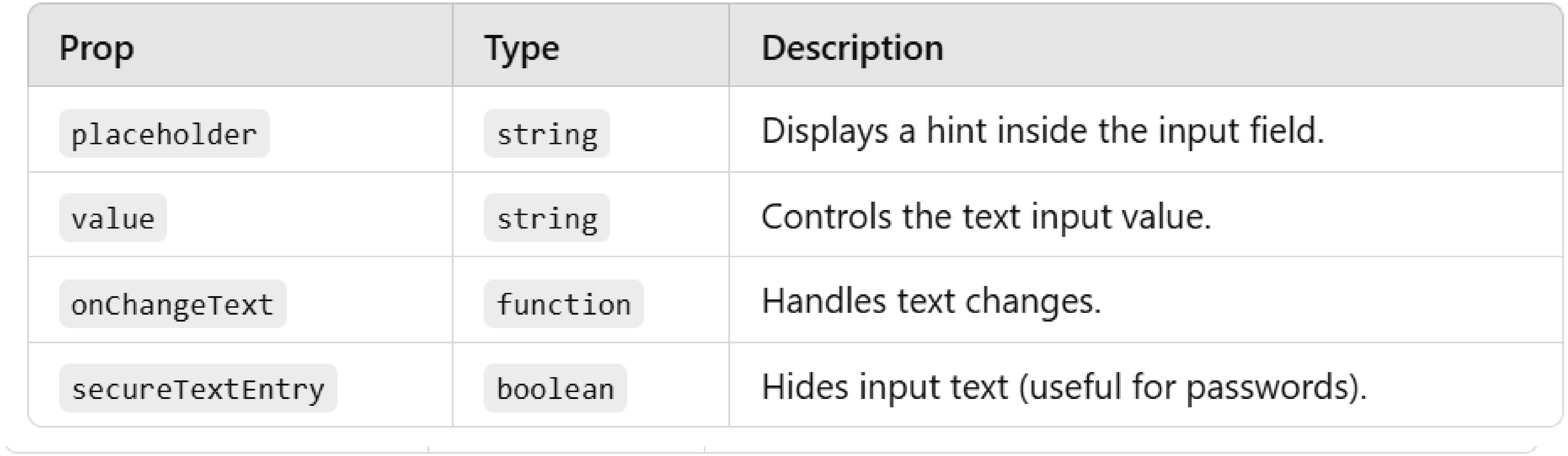
Prop	Type	Description
`size`	`"small"`, `"large"`, `number`	Specifies the size of the spinner.
`color`	`string`	Changes the color of the spinner.
`animating`	`boolean`	Controls whether the indicator is visible.

Core component: `ActivityIndicator`

```
import { ActivityIndicator, View } from 'react-native';

export default function App() {
    return (
        <View>
            <ActivityIndicator
                size="large"
                color="#0000ff"
                animating={true} />
        </View>
    );
}
```

Core component: `TextInput`

- `TextInput` allows users to enter text input, such as searching for a city.
- Some props:



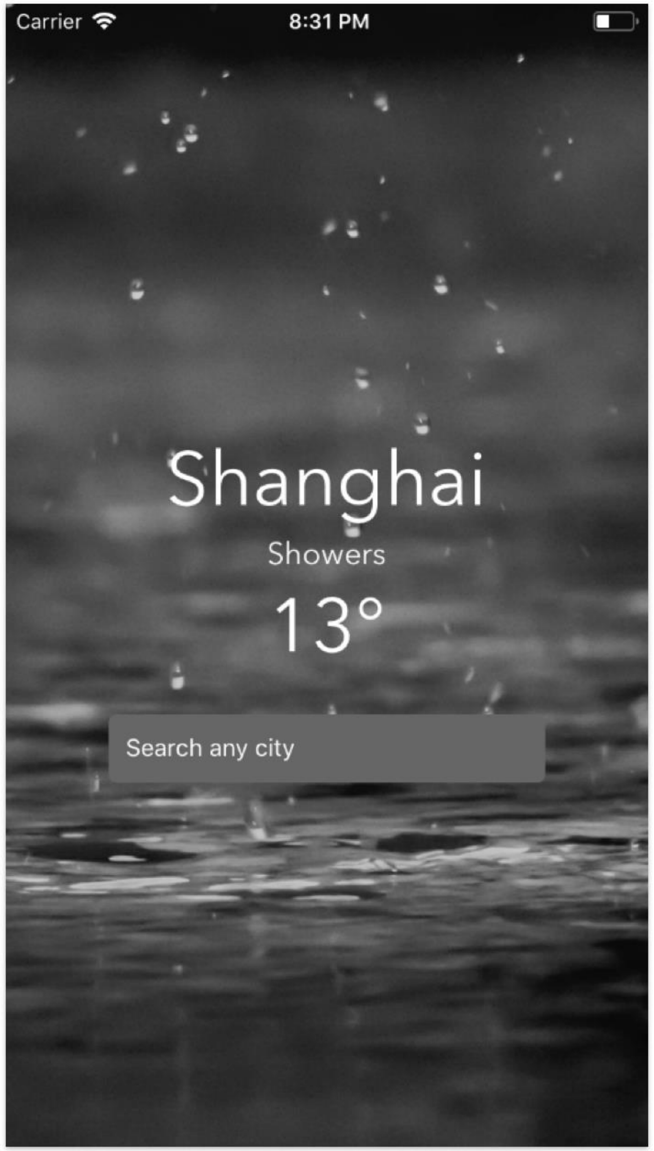
Prop	Type	Description
`placeholder`	`string`	Displays a hint inside the input field.
`value`	`string`	Controls the text input value.
`onChangeText`	`function`	Handles text changes.
`secureTextEntry`	`boolean`	Hides input text (useful for passwords).

Core component: `TextInput`

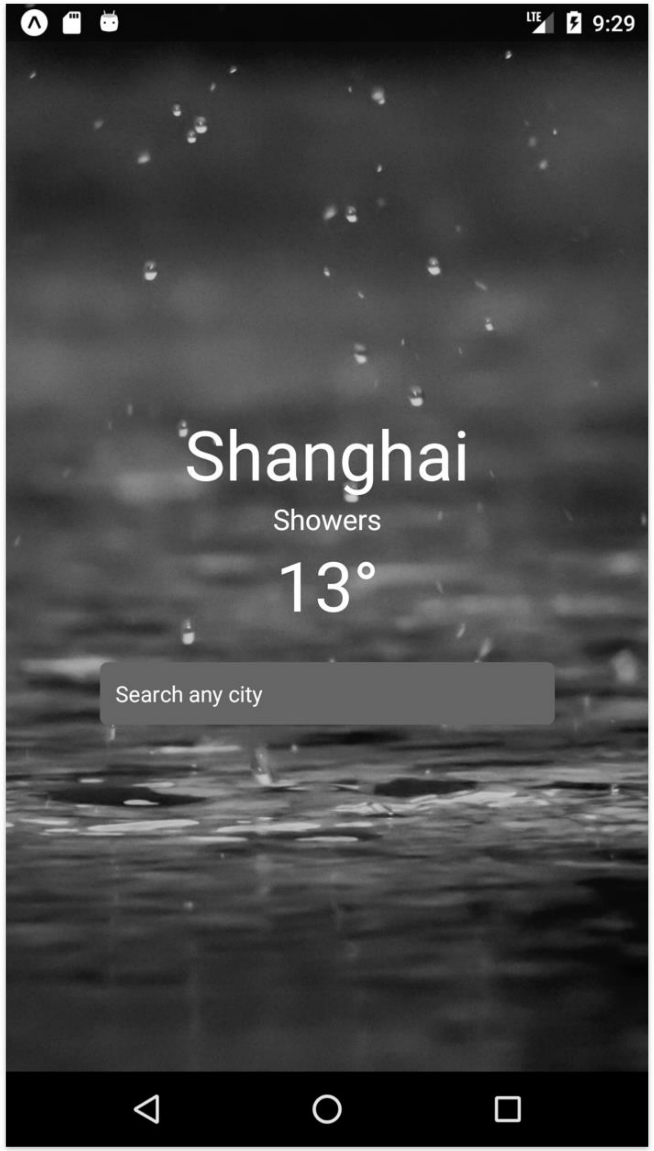
```
export default function AppTextInput() {
    return (
        <TextInput
            style={{
                padding: 10,
                borderWidth: 1,
                borderColor: '#ccc',
                margin: 10
            }}
            placeholder="Enter city name"
            onChangeText={(text) => console.log(text)}
        />
    );
}
```

The finished Weather App's look

iOS



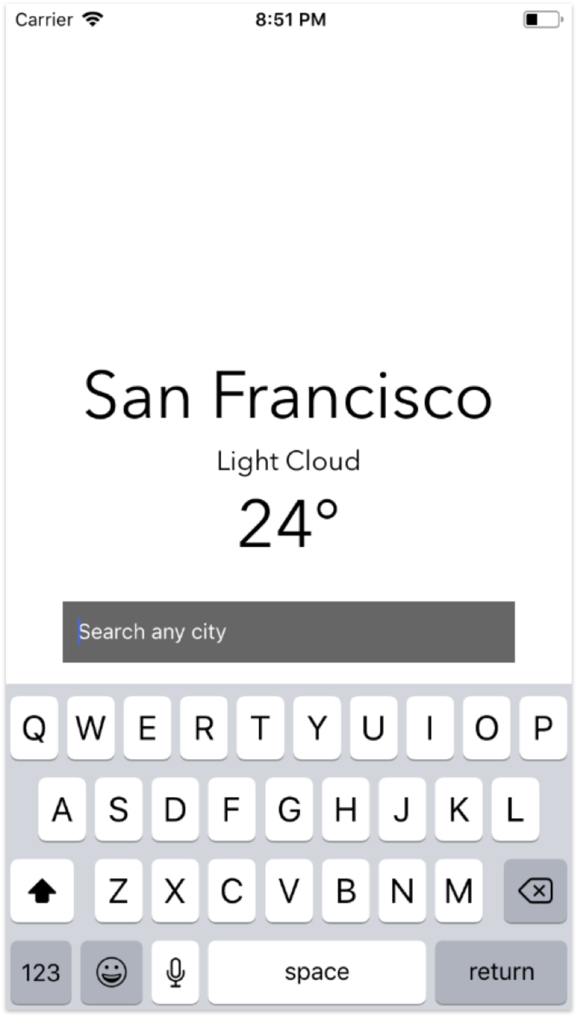
Android



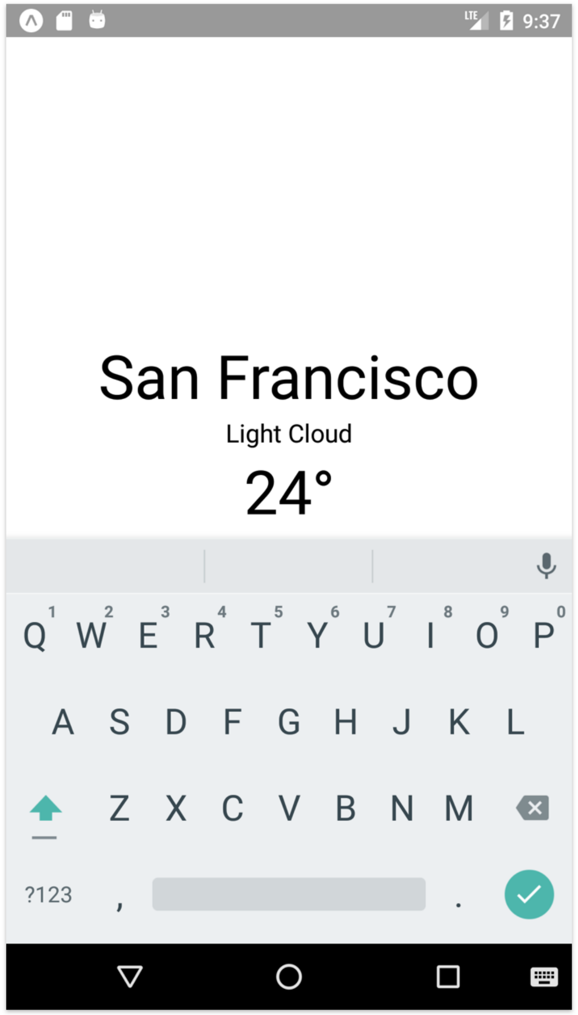
Layout problem with virtual keyboard

- ❖ The virtual keyboard can cover roughly half the device screen
- ➢ React Native provides the `KeyboardAvoidingView` to solve this problem
  - Use this component instead of the normal `View` component
- 3 behaviors: `height`, `padding`, `position` (please test 'em out!)

iOS

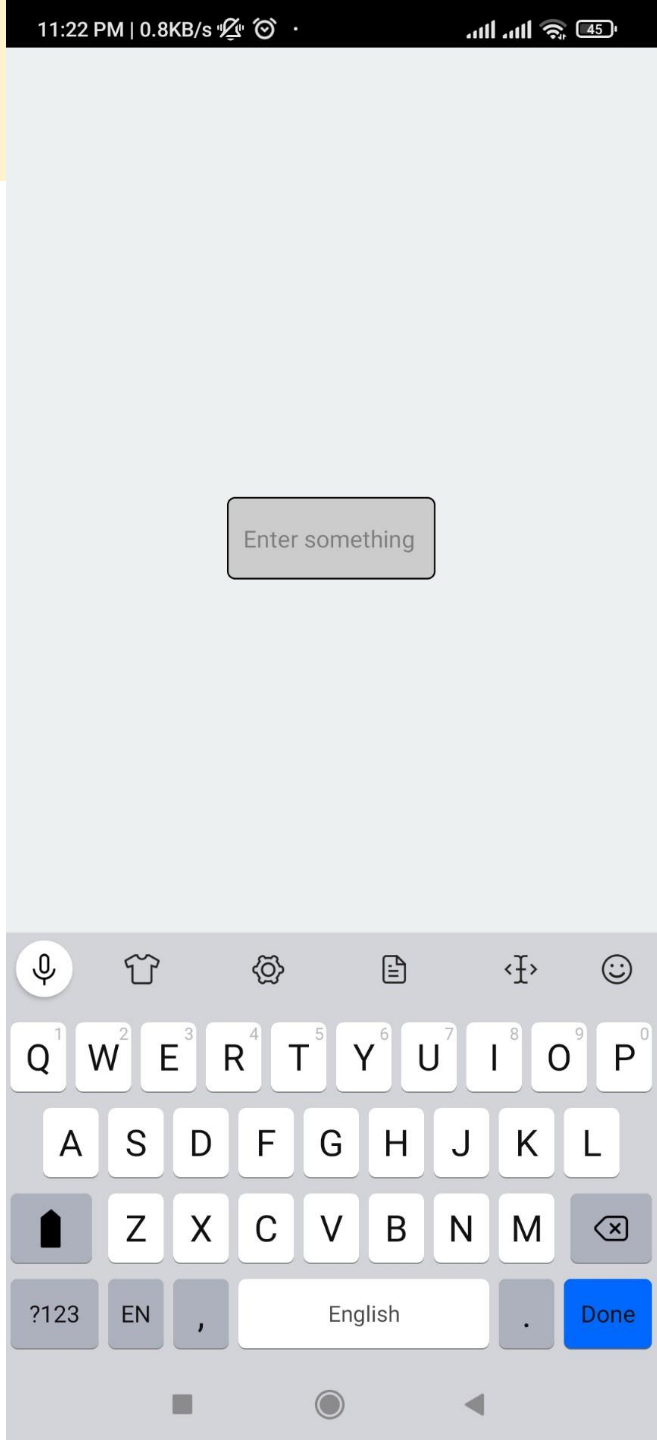


Android



KeyboardAvoidingView example

```
<KeyboardAvoidingView style={{
    flex: 1,
    alignItems: 'center',
    justifyContent: 'center',
    backgroundColor: '#ECF0F1'
}} behavior="height">
    <StatusBar barStyle="light-content" />
    <TextInput style={{
        padding: 10,
        borderWidth: 1,
        borderRadius: 5,
        backgroundColor: '#ccc'
    }} placeholder='Enter something' />
</KeyboardAvoidingView>
```



Calling APIs in React Native

- APIs (Application Programming Interfaces) allow mobile applications to communicate with external servers to fetch or send data.
- In React Native, we can use the `fetch()` method to call APIs, retrieve weather data, and update the UI dynamically.
- The `fetch()` function is a built-in JavaScript method used to make HTTP requests to an API.
  - It returns a Promise, which allows us to handle asynchronous operations effectively.

Calling APIs in React Native

- Using `fetch()` for API calls
  - Example API URL for weather information:
    `https://api.open-meteo.com/v1/forecast?latitude=35&longitude=139&current_weather=true`

```
const getWeather = async () => {
    try {
        const response = await fetch("API_URL");
        const data = await response.json();
        console.log(data);
    } catch (error) {
        console.error("Error fetching weather:", error);
    }
};
```

Call `getWeather()` when the app loads:

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

const App = () => {
    useEffect(() => {
        getWeather();
    }, []);
    return null;
};
```

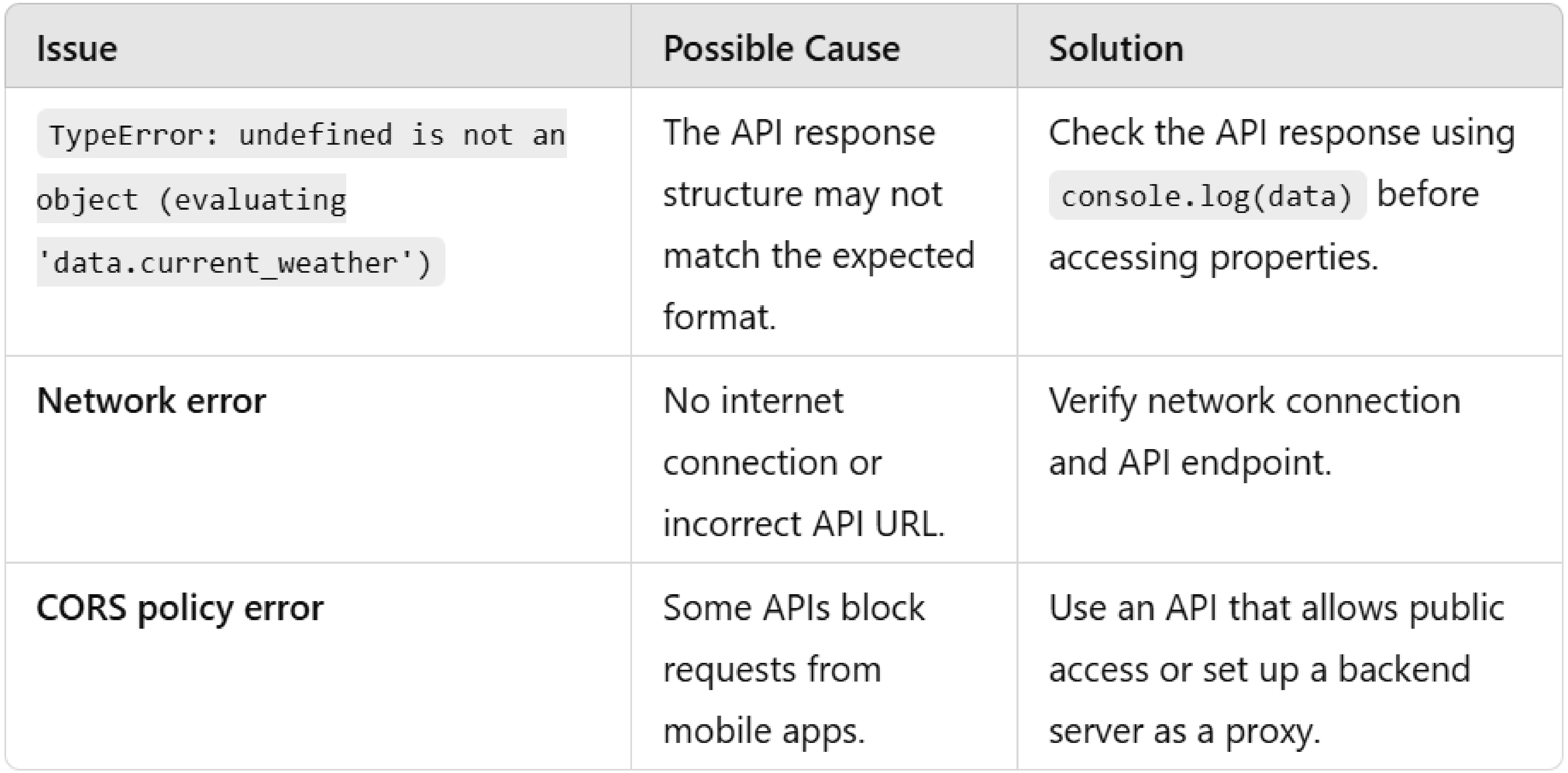
Store API data in state using `useState`

```
export default WeatherApp = () => {
    const [weather, setWeather] = useState(null);
    const fetchWeather = async () => {
        try {
            const response = await fetch("API_URL");
            const data = await response.json();
            setWeather(data.current_weather);
        } catch (error) {
            console.error("Error fetching weather:", error);
        }
    };
    useEffect(fetchWeather, []);
}
```

Store API data in state using `useState`

```
export default WeatherApp = () => {
    // omitted code
    return (
        <View>
            {
                weather ?
                    <Text>Temperature: {weather.temperature}°C</Text>
                    :
                    <Text>Loading...</Text>
            }
        </View>
    );
};
```

Common Issues & Solutions



Issue	Possible Cause	Solution
`TypeError: undefined is not an object (evaluating 'data.current_weather')`	The API response structure may not match the expected format.	Check the API response using `console.log(data)` before accessing properties.
Network error	No internet connection or incorrect API URL.	Verify network connection and API endpoint.
CORS policy error	Some APIs block requests from mobile apps.	Use an API that allows public access or set up a backend server as a proxy.

Weather API

- URL
  - `https://api.open-meteo.com/v1/forecast`
- Input parameters used by our app

• `latitude`, `longitude`	The location of weather forecast
• `current_weather`	Set this to true to get current weather

- Example URL

```
https://api.open-meteo.com/v1/forecast
?latitude=21.02&longitude=105.84&current_weather=true
```

- How do we get the latitude and longitude of a City?

GeoCoding API

- Example URL

  `https://geocoding-api.open-meteo.com/v1/search?name=Thanh%20Xuan`

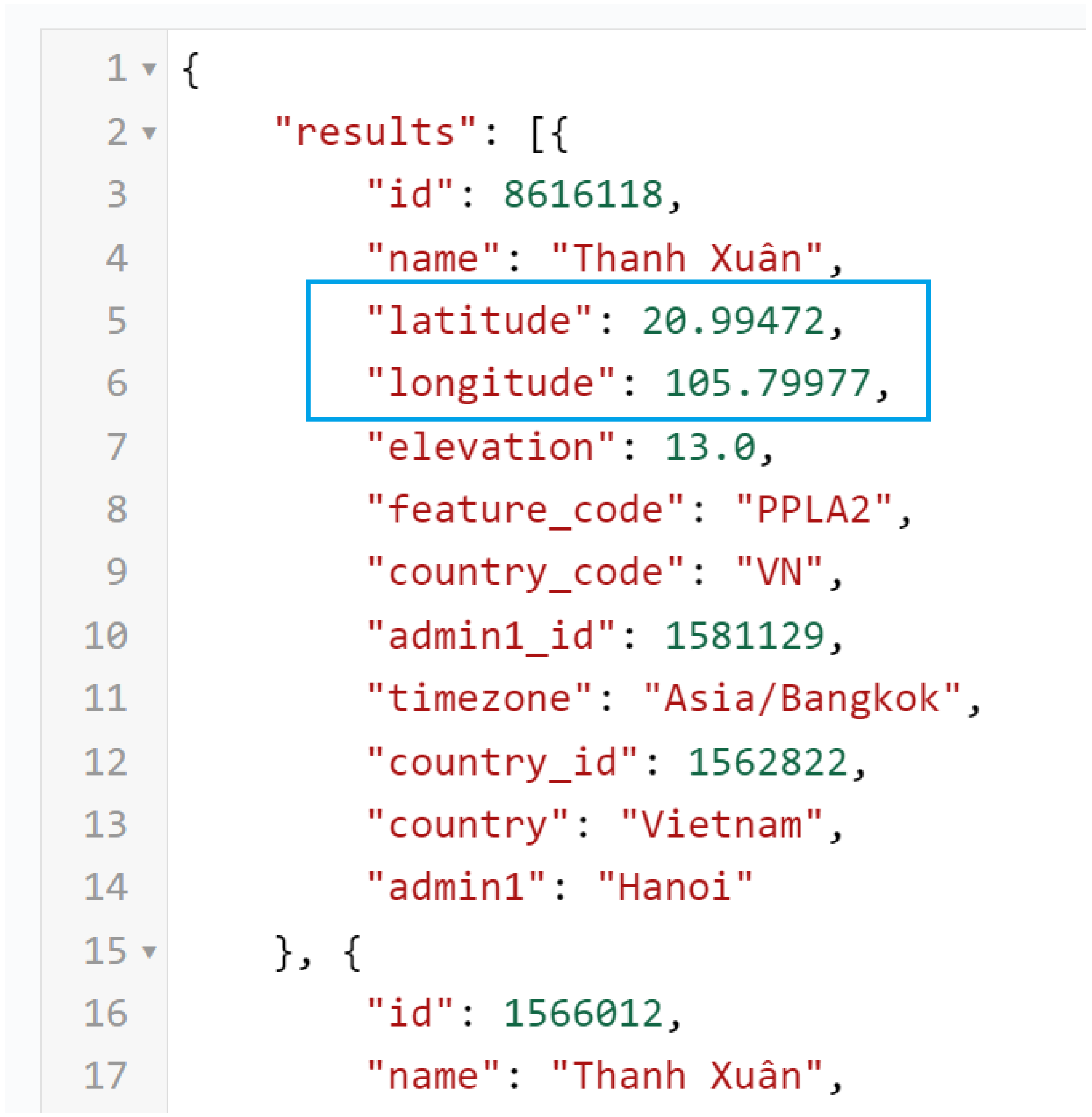
- Input parameters used by our app

• `name`	The name of the place

- Open the above URL on browser to see the format of output
  - Use https://jsonlint.com or similar tool to format the JSON string to make it readable

GeoCoding API Output

- Output is an array of locations stored in the `results` property
- We need the *first* one (indexed 0)
- We are interested in the `latitude` and `longitude` properties



```
{
    "results": [{
        "id": 8616118,
        "name": "Thanh Xuân",
        "latitude": 20.99472,
        "longitude": 105.79977,
        "elevation": 13.0,
        "feature_code": "PPLA2",
        "country_code": "VN",
        "admin1_id": 1581129,
        "timezone": "Asia/Bangkok",
        "country_id": 1562822,
        "country": "Vietnam",
        "admin1": "Hanoi"
    }, {
        "id": 1566012,
        "name": "Thanh Xuân",
const interpretWeather = (code) => {
    if (code <= 1)
        return 'Clear sky';
    if (code > 1 && code <= 3)
        return 'Partly cloudy';
    else if (code == 45 || code == 48)
        return 'Fog';
    else
        return 'Unknown: ' + code;
};
```

WMO Weather interpretation codes (WW)

Code	Description
0	Clear sky
1, 2, 3	Mainly clear, partly cloudy, and overcast
45, 48	Fog and depositing rime fog
51, 53, 55	Drizzle: Light, moderate, and dense intensity
56, 57	Freezing Drizzle: Light and dense intensity
61, 63, 65	Rain: Slight, moderate and heavy intensity
66, 67	Freezing Rain: Light and heavy intensity
71, 73, 75	Snow fall: Slight, moderate, and heavy intensity
77	Snow grains
80, 81, 82	Rain showers: Slight, moderate, and violent
85, 86	Snow showers slight and heavy
95 *	Thunderstorm: Slight or moderate
96, 99 *	Thunderstorm with slight and heavy hail