React Native APIs & Fetching Data: Beginner to Advanced

Subtitle: Learn how to retrieve and manage data from external APIs in your React Native apps.

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

Introduction

Most apps need to interact with external data sources like APIs. React Native makes it easy to fetch, display, and manage data. This guide covers fetching data using `fetch` and `axios`, handling asynchronous operations, and updating the UI dynamically.

Step 1: Fetching Data with Fetch API

React Native supports the **native `fetch` function** to request data from APIs.

Example:

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

export default function App() {
  const [data, setData] = useState([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    fetch('https://jsonplaceholder.typicode.com/posts')
      .then(response => response.json())
      .then(json => setData(json))
      .catch(error => console.error(error))
      .finally(() => setLoading(false));
  }, []);

  if (loading) return <ActivityIndicator size="large" color="#0000ff" />;

  return (
    <View style={{ flex: 1, padding: 20 }}>
      <FlatList
        data={data}
        keyExtractor={item => item.id.toString()}
```

```
        renderItem={({ item }) => <Text>{item.title}</Text>}
      />
    </View>
  );
}
```

Exercise: Fetch data from `https://jsonplaceholder.typicode.com/users` and display user names in a list.

Step 2: Fetching Data with Axios

Axios is a popular library that simplifies HTTP requests.

1. Install Axios:

```
npm install axios
```

2. Example usage:

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

export default function App() {
  const [users, setUsers] = useState([]);

  useEffect(() => {
    axios.get('https://jsonplaceholder.typicode.com/users')
      .then(response => setUsers(response.data))
      .catch(error => console.error(error));
  }, []);

  return (
    <View style={{ flex: 1, padding: 20 }}>
      <FlatList
        data={users}
        keyExtractor={item => item.id.toString()}
        renderItem={({ item }) => <Text>{item.name}</Text>}
      />
    </View>
  );
}
```

Exercise: Convert the previous `fetch` example to use Axios instead.

Step 3: Handling Loading and Errors

- Always use **loading indicators** while fetching data.
- Use **try/catch** or `.catch()` to handle errors.
- Display friendly messages for API failures.

Example:

```
const fetchData = async () => {
  try {
    setLoading(true);
    const response = await axios.get('https://jsonplaceholder.typicode.com/posts
    setData(response.data);
  } catch (error) {
    console.error(error);
    alert('Failed to fetch data');
  } finally {
    setLoading(false);
  }
};
```

Step 4: Updating State Dynamically

- Use `useState` to store API data.
- Update state after API calls to **re-render UI**.
- For more complex apps, consider **useReducer** or **state management libraries**.

Exercise: Build a **Post Viewer app** that fetches posts and displays them in a scrollable list with titles and body content.

Step 5: Best Practices

- Always handle **loading and error states**.
- Keep **API calls in `useEffect`** to avoid repeated requests.
- Use **FlatList** for rendering long lists efficiently.

- Consider **caching data** for offline scenarios.
- Use **async/await** for clean and readable code.

Fetching and managing API data is critical for dynamic apps. Mastering these techniques allows you to build apps that **interact with real-world data** and provide engaging user experiences.

Visit **haas.dev** for more React Native guides, tutorials, and project examples.

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