Pavneet Singh

Fetch and Populate XML Data in React Native

Pavneet Singh

Sep 15, 2020 • 7 Min read • 2,856 Views

Sep 15, 2020 • 7 Min read • 2,856 Views

Web Development Front End Web Dev... Client-side Framew... React

Introduction

1

Introduction

XML (Extensible Markup Language) is one of the famous formats for encoding documents and data using markup tags. XML wraps every piece of data inside specific customized key tags with values:

XML

```xml
<person>
  <fname>Pavneet</fname> <!--first name-->
  <lname>Singh</lname> <!--last name-->
</person>
```

XML documents are widely used in Android to create UI layouts, which can be used to provide native interface support via native modules in React Native. XML is also the only supported data transfer format for APIs that are built using SOAP protocol.

This guide provides the implementation details to download and parse XML documents in React Native.

Installing Dependencies

The basic requirements to implement an API request can be categorized into three key steps: data source, network API module, and parsing API module (optional):

Set Up Data Source

The first step is to set up a mock XML response data, which can be done easily by creating a GitHub gist. Create a gist with one XML response file like this.

> ***Note***
> - *A gist can be copied and used with any GitHub account. Any change in a gist file will produce a new raw URL.*

> - *A custom mock response can also be created using mocky. Alternately, you can use the gorest mock API for fake posts XML data.*

Network Module

React Native provides an inbuilt `fetch` API to make network calls so there is no need to install any other dependencies. Alternatively, there are other HTTP clients also available, like `XMLHttpRequest` (inbuilt), `Frisbee`, `axios`, `superagent`, and `request`.

> *The current version of `axios` tries to automatically convert the response into the JSON object, but this behavior could change in the future.*

Parsing Module

React Native doesn't offer any inbuilt modules for parsing XML data, so `fast-xml-parser` will be used to parse XML response. Install it as a dependency:

```sh
npm install fast-xml-parser typescript
```

> *The `typescript` package is required to fix the type warning.*

There are many other third-party solutions available but they have some drawbacks:

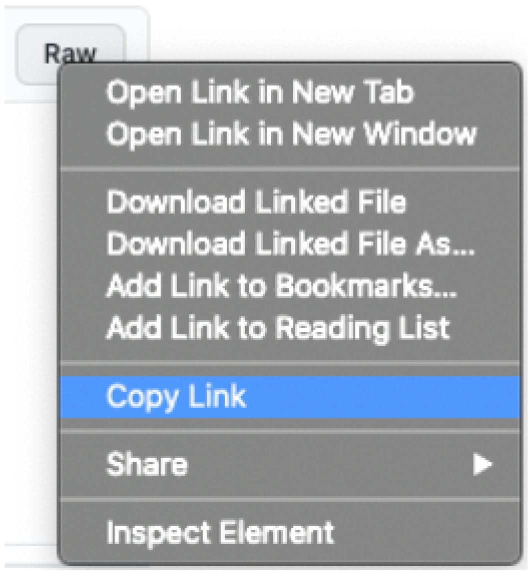
- `xml2js` is one of the popular library for parsing XML but it requires additional dependencies (event, timer, etc.) to work with React Native.
- `jsdom` is a fully featured DOM creation and parsing lib, mostly suitable for DOM manipulation.
- `xmldom` is a W3C standard based paring library, but at the time of this writing it didn’t work well with comments, and gives warnings for unclosed tags due to XML comments.

Also, there are some React Native-specific libraries available:

- react-native-xml2js is a React library specifically modified for React Native. It’s no longer being maintained, but it can be used as a last resort.
- react-xml-parser is a lightweight XML parser, but it’s limited to the `getElementsByTag` function and does not work well with XML comments.

Downloading XML Response

To download the XML data from a gist, copy the `raw` URL of the GitHub gist by right-clicking on the `raw` button and copy the link:



A `fetch` API call can simply be executed by using `fetch(input: RequestInfo)` where `RequestInfo` can be a string URL:

JSX

```
fetch('https://gist.githubusercontent.com/Pavneet-Sing/d0f3324f2cd3244a6ac
    .then((response) => response.text())
    .then((textResponse) => console.log('response is ', textResponse))
    .catch((error) => {
        console.log(error);
    });
```

The above `fetch` call will return a `Promise` that is combined with a `then` call to process the asynchronous response. The `text()` function call also returns the `Promise` object that is combined with another `then` function to log the text response. The above code can be simplified using `async/await`:

JSX

```jsx
    async getXMLResponse() {
        const response = await fetch('https://gist.githubusercontent.com/Pavne
        console.log('response is', await response.text());
    }
```

> *`async/await` do not handle errors, so a `try/catch` block should be used to handle any potential errors from an `await` execution.*

Parsing XML Data

The `fast-xml-parser` library provides a convenient way to parse the string XML response using the `parse` function. The `parse` function will convert the XML response text into a `JSON` object, which allows direct access to the properties of a parsed JSON object to get the data:

JSX

```jsx
    import { parse } from 'fast-xml-parser';

    getXMLResponse = () => {
        fetch('https://gist.githubusercontent.com/Pavneet-Sing/d0f3324f2cd324
        .then((response) => response.text())
        .then((textResponse) => {
          let obj = parse(textResponse);
          let fname = obj.person.fname;
          let lname = obj.person.lname;
          let phone = obj.person.contacts.personal.phone;
          this.setState({ fname: fname, lname: lname, phone: phone })
        })
        .catch((error) => {
          console.log(error);
        });
    }
```

The `parse(textResponse)` function call will convert the XML text response to an object, and the data will be extracted using the properties' names like `obj.person.fname`. The parsed data is being stored in the `state` using `setState` function to display the response on the UI. The optimized React Native code to display the data on the UI is available on my RNParseXML repository.

Conclusion

Parsing `XML` into a JSON object offers great flexibility to access the data directly with properties. Objects can easily be traversed to parse nested arrays or objects. React Native offers many inbuilt

React APIs (like `fetch`, `async/await`, etc.) but does not support all the React libraries, so it's recommended to always test the implementation on Android and iOS platforms. Happy coding!

1