Handing JSON Data Errors in Slim 3

Rob Allen shows us how to handle JSON data errors in Slim 3.

by Rob Allen MVB · Nov. 28, 16 · Web Dev Zone · Tutorial

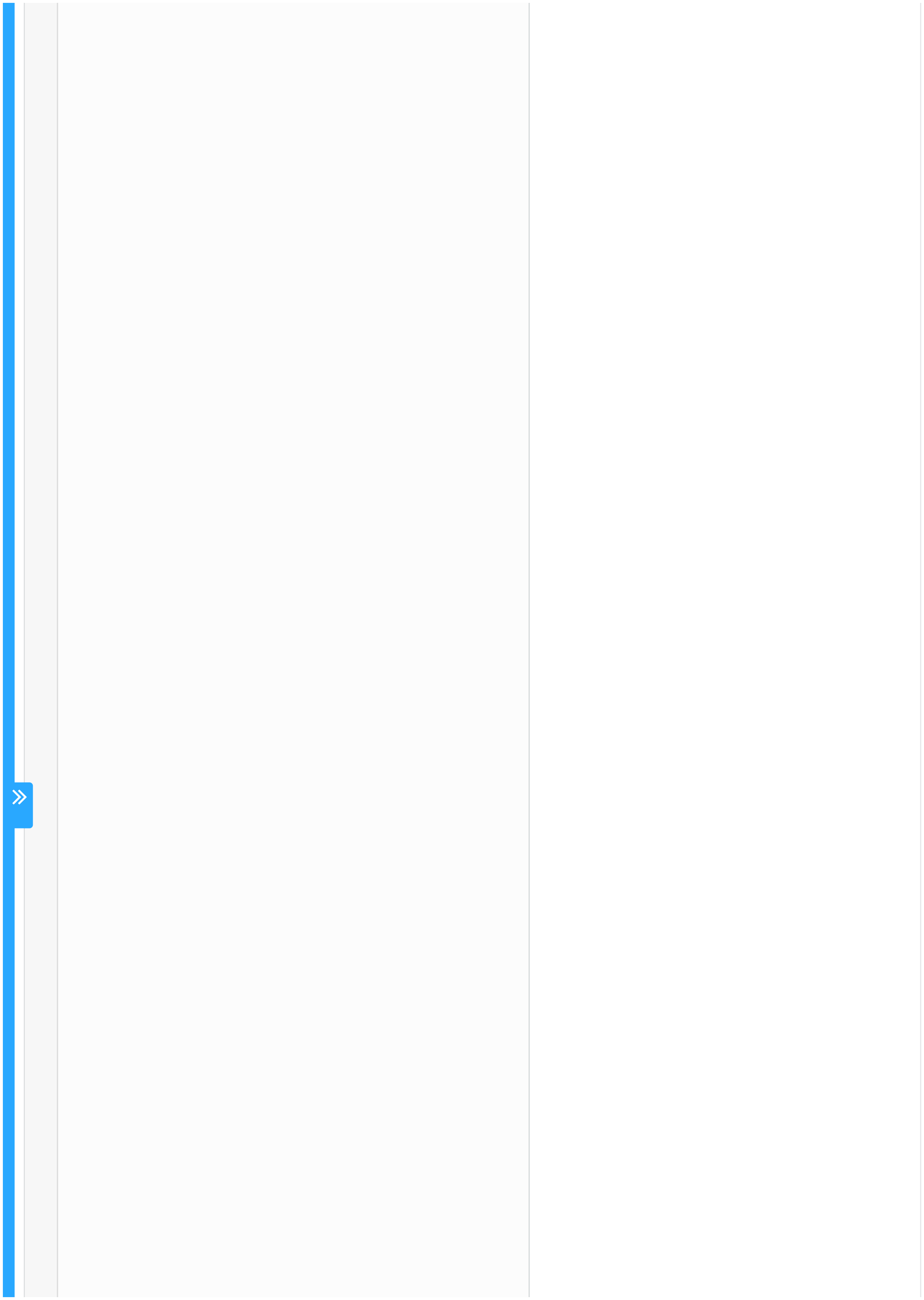
When you send JSON data into a Slim Framework application with a content-type of application/json, then Slim will decode it for you if you use getParsedBody():

```
$app->post("/", function ($request, $response, $args) {
$input = $request->getParsedBody();

var_dump($input);exit;
});
```

Using curl to test:

```
$ curl -H "Content-Type: application/json" http://localhost:8888 -d '{"foo": "bar"}'
array(1) {
'foo' =>
string(3) "bar"
}
```



If there's an error however, you get this:

```
$ curl -H "Content-Type: application/json" http://localhost:8888 -d '{foo: bar}'
NULL
```

If you care about this, you can use json_last_error_msg() and return an error:

```
$app->post("/", function ($request, $response, $
args) {
$input = $request->getParsedBody();
if ($input === null) {
```

```
return $response->withJson(
['error_decoding_json' => json_last_error_msg()]
,
400,
JSON_PRETTY_PRINT
);
}

var_dump($input);exit;
});
```

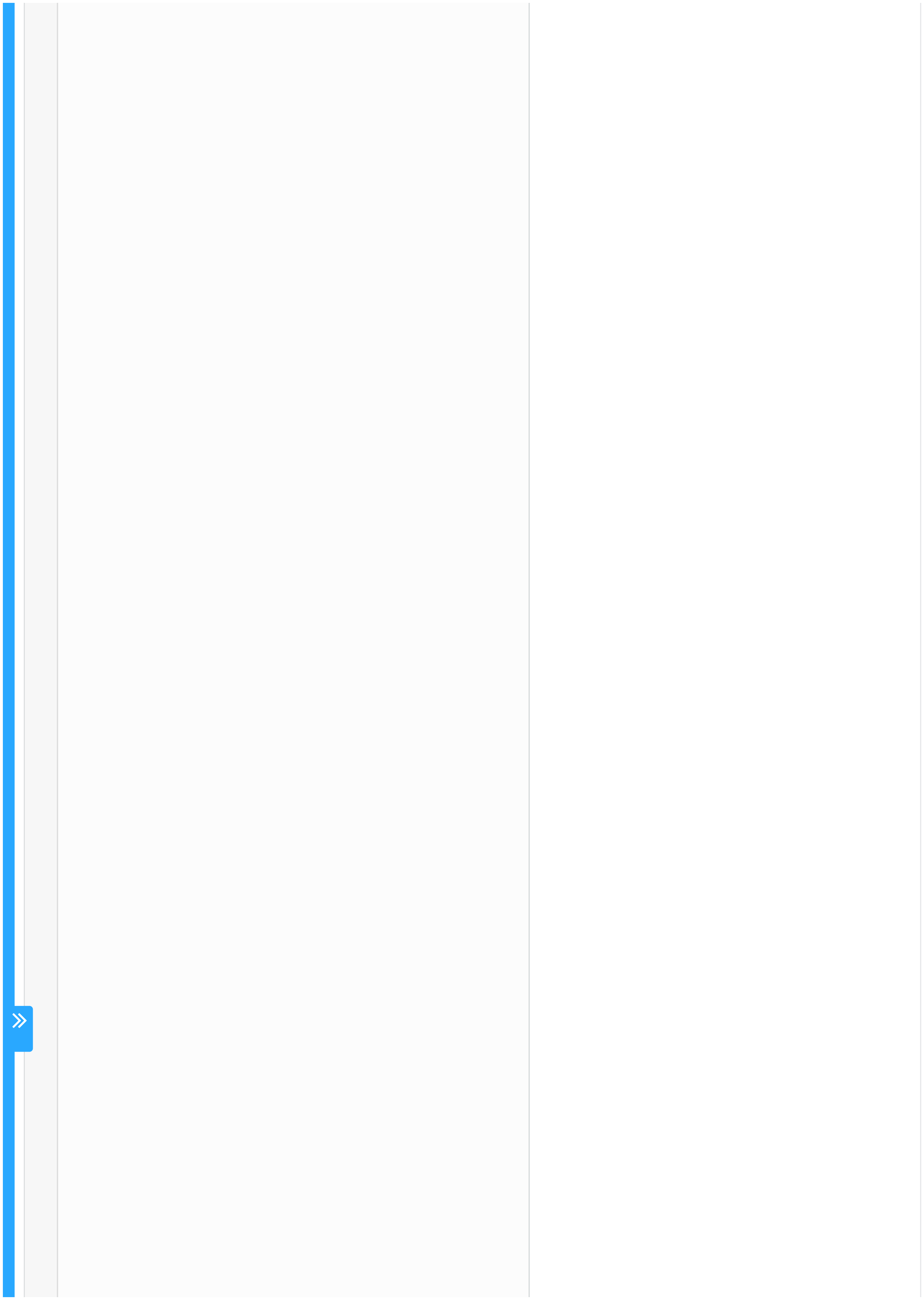
(note – in real code, you should check that the Accept header was a JSON one…)

Don't forget the JSON_PRETTY_PRINT as a human is going to be reading this error, so make it easier for them.

Use jsonlint for More Information

If you really want to provide great diagnostics, then use jsonlint:

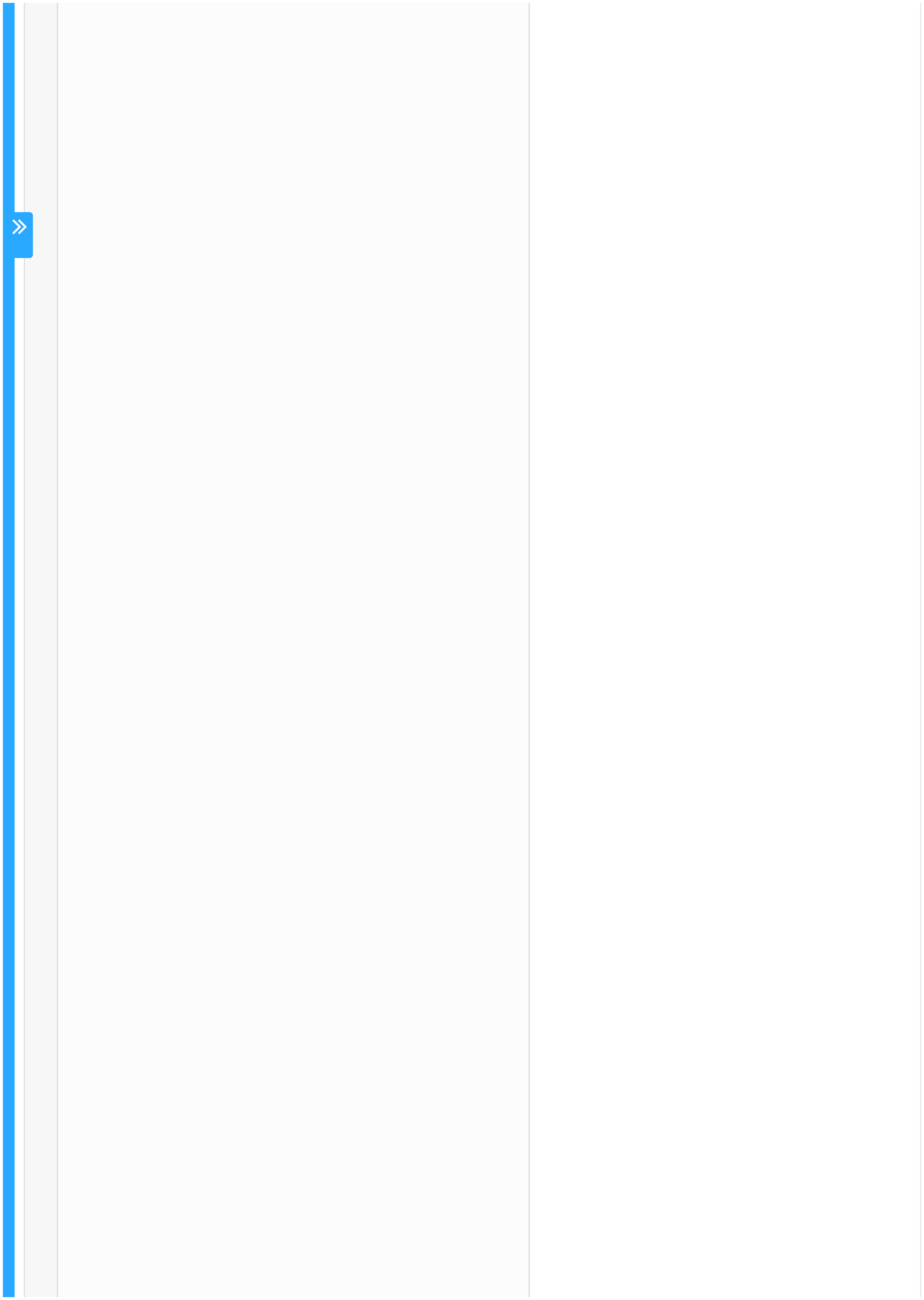
```
$ composer require seld/jsonlint
```



Update your handler like this:

```php
$app->post("/", function ($request, $response, $args) {
$input = $request->getParsedBody();
if ($input === null) {
$parser = new \Seld\JsonLint\JsonParser();
$result = $parser->lint($input);
if ($result instanceof \Seld\JsonLint\ParsingException) {
return $response->withJson(
['error_decoding_json' => $result->getMessage()],
400,
JSON_PRETTY_PRINT
);
}
}

var_dump($input);exit;
});
```



(lint() will return NULL or a ParsingException, so we don't need to test for anything else.)

The result looks like this:

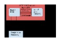
```
$ curl -H "Content-Type: application/json" http://localhost:8888 -d '{foo: bar}'
{
"error_decoding_json": "Parse error on line 1:\n\n^\nExpected one of: 'STRING', 'NUMBER', 'NULL', 'TRUE', 'FALSE', '{', '['"
}
```

This is much more informative!

Scripting as a Service, SPA Support. Watch on-demand now.

Like This Article? Read More From DZone

POSTing JSON Data with PHP cURL


CURL Comes to N1QL: Querying External JSON Data

Use Curl to Create a CouchDB Admin User

Free DZone Refcard
Low-Code Application Development

Topics: INPUT , CURL , DIAGNOSTICS , SLIM , JSON

Like (0) Comment (2) Save
Tweet 4,607 Views

Published at DZone with permission of Rob Allen , DZone MVB. See the original article here.
Opinions expressed by DZone contributors are their own.

Web Dev Partner Resources

ONLYOFFICE code samples: see how editors will work within your app
OnlyOffice

The Ultimate 36-Point Checklist for Choosing a Headless CMS (eBook)
DotCMS

Detailed documentation for ONLYOFFICE integration
OnlyOffice

Bring collaborative document editing to your web app with ONLYOFFICE
OnlyOffice

HERE topples Google to take first place
HERE

From Brick and Mortar to Bits and Bytes: Why Cognitive Services are Transforming eCommerce
PubNub

A World Transformed: Building Smarter, Next Generation Apps with Cognitive Services
PubNub

Welcome to the Cognitive Era - The New Generation of Computing
PubNub

Containers 101: Everything You Need to Know About Containers [eBook]
DotCMS

HERE ranks #1 for developers against Google and Mapbox
HERE

Building & Managing Single Page Apps in

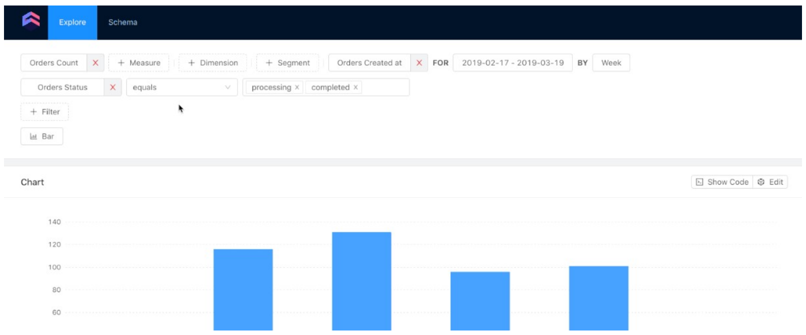
React Query Builder With Cube.js

In this post, we look at how we can use these two open source libraries to create a query functionality in an application using JavaScript.

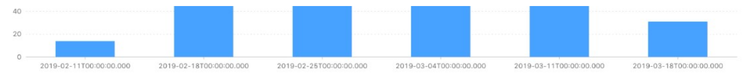
by Artyom Keydunov · Mar 28, 19 · Web Dev Zone · Tutorial

Starting from version 0.4, the React Cube.js client comes with the `<QueryBuilder />` component. It is designed to help developers build interactive analytics query builders. The `<QueryBuilder />` abstracts state management and API calls to the Cube.js backend. It uses the render prop and doesn't render anything itself, but calls the render function instead. This way it gives maximum flexibility to building a custom-tailored UI with a minimal API.

The example below shows the `<QueryBuilder />` component in action with Ant Design UI framework elements.



Orders Count | Orders Created at FOR 2019-02-17 - 2019-03-19 BY Week
Orders Status | equals | processing completed



The above example is from Cube.js Playground. You can check its source code on GitHub.

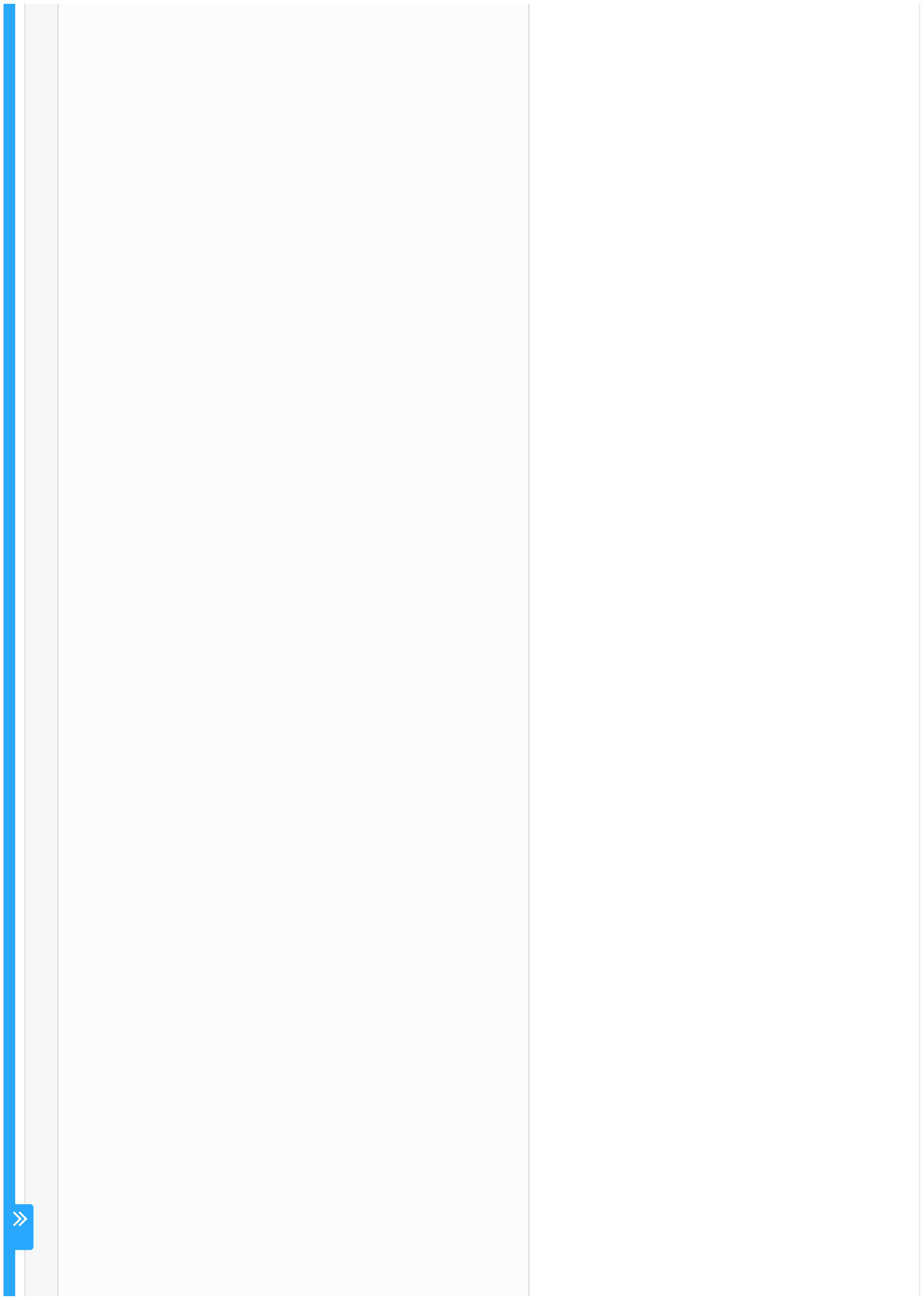
This tutorial walks through building the much simpler version of the query builder. But it covers all the basics you need to build one of your own.

Setup a Demo Backend

If you already have Cube.js backend up and running you can skip this step.

First, let's install the Cube.js CLI and create a new application with a Postgres database.

```
$ npm install -g cubejs-cli
$ cubejs create -d postgres react-query-builder
```



We host a dump with sample data for tutorials. It is a simple “E-commerce database” with orders, products, product categories, and users tables.

```
$ curl http://cube.dev/downloads/ecom-dump.sql >
ecom-dump.sql
$ createdb ecom
$ psql --dbname ecom -f ecom-dump.sql
```

Once you have data in your database, change the content of the `.env` file inside your Cube.js directory to the following. It sets the credentials to access the database, as well as a secret to generate auth tokens.

```
CUBEJS_DB_NAME=ecom
CUBEJS_DB_TYPE=postgres
CUBEJS_API_SECRET=SECRET
```

Now that we have everything configured, the last step is to generate a Cube.js schema based on some of our tables and start the dev server.

```
$ cubejs generate -t line_items
$ yarn dev
```

If you open *http://localhost:4000* in your browser you will access the Cube.js Playground. It is a development environment which generates the Cube.js schema, creates scaffolding for charts, and more. It has its own query builder which lets you generate charts with different charting libraries.

Now, let's move on to building our own query builder.

Building a Query Builder

The `<QueryBuilder />` component uses the render props technique. It acts as a data provider by managing the state and API layer and calls `render` props to let developers implement their render logic.

Besides `render`, the only required prop is `cubejsApi`. It expects an instance of your cube.js API client returned by the `cubejs` method.

Here you can find a detailed reference of the `<QueryBuilder />` component.

```
import cubejs from "@cubejs-client/core";
import { QueryBuilder } from "@cubejs-client/react";
const cubejsApi = cubejs("CUBEJS_TOKEN", { apiurl: "CUBEJS_BACKEND_URL" });

export default () => (
<QueryBuilder
cubejsApi={cubejsApi}
render={queryBuilder => {
// Render whatever you want based on the state of queryBuilder
}}
/>
);
```

The properties of `queryBuilder` can be split into categories based on what element they are referred to. To render and update measures, you need to use `measures`, `availableMeasures`, and `updateMeasures`.

`measures` is an array of already selected measures. It is usually empty in the beginning (unless you passed a default `query` prop). `availableMeasures` is an array of all measures loaded via API from your Cube.js data schema. Both `measures` and `availableMeasures` are arrays of objects with `name`, `title`, `shortTitle`, and `type` keys. `name` is used as an ID. `title` could be used as a human-readable name, and `shortTitle` is only the measure's title without the Cube's title.

```
// `measures` and `availableMeasures` are arrays with the following structure
[
{ name: "Orders.count", title: "Orders Count", shortTitle: "Count", type: "number" },
{ name: "Orders.number", title: "Orders Number", shortTitle: "Number", type: "number" }
]
```

`updateMeasures` is an object with three functions: `add`, `remove`, and `update`. It is used to control the state of the query builder related to measures.

Now, using these properties, we can render a UI to manage measures and render a simple line chart, which will dynamically change the content based on the state of the query builder.

```
import React from "react";
import ReactDOM from "react-dom";
import { Layout, Divider, Empty, Select } from "antd";
import { QueryBuilder } from "@cubejs-client/react";
import cubejs from "@cubejs-client/core";
import "antd/dist/antd.css";

import ChartRenderer from "./ChartRenderer";

const cubejsApi = cubejs(
"YOUR-CUBEJS-API-TOKEN",
{ apiUrl: "http://localhost:4000/cubejs-api/v1" }
);

const App = () => (
<QueryBuilder
query={{
timeDimensions: [
{
dimension: "LineItems.createdAt",
granularity: "month"
}
]
}}
cubejsApi={cubejsApi}
render={({ resultSet, measures, availableMeasures, updateMeasures }) => (
<Layout.Content style={{ padding: "20px" }}>
<Select
mode="multiple"
style={{ width: "100%" }}
placeholder="Please select"
onSelect={measure => updateMeasures.add(measure)}
onDeselect={measure => updateMeasures.remove(measure)}
>
{availableMeasures.map(measure => (
<Select.Option key={measure.name} value={measure}>
{measure.title}
</Select.Option>
))}
</Select>
<Divider />
{measures.length > 0 ? (
<ChartRenderer resultSet={resultSet} />
) : (
<Empty description="Select measure or dimension to get started" />
)}
</Layout.Content>
)}
/>
);

const rootElement = document.getElementById("root");
ReactDOM.render(<App />, rootElement);
```

The code above is enough to render a simple query builder with a measure select. Here’s how it looks in the CodeSandbox:

Similar to `measures`, `availableMeasures`, and `updateMeasures`, there are properties to render and manage dimensions, segments, time, filters, and chart types. You can find the full list of properties in the documentation.

Also, it is worth checking the source code of a more complicated query builder from Cube.js Playground. You can find it on GitHub here.

Like This Article? Read More From DZone

Declarative D3 Charts With React 16.3 [Text and Video]

PyDev of the Week: Jason Myers

4 Useful JavaScript Libraries for Data Analysis and Visualization

Free DZone Refcard
Low-Code Application Development

Topics: WEB DEV, REACT.JS, CUBE.JS, DATA VISUALIZATION

Like (1) Comment (0) Save
Tweet 627 Views