Default Route Arguments in Slim

Interested in learning the basics of this Ruby template engine? In this article, we show you how to set up a Hello World message using Slim.

by Rob Allen MVB · Sep. 04, 17 · Web Dev Zone · Tutorial

A friend of mine recently asked how to do default route arguments and route specific configuration in Slim, so I thought I'd write up how to do it.

Consider a simple Hello route:

```
$app->get("/hello[/{name}]", function ($request, $response, $args) {
$name = $request->getAttribute('name');
return $response->write("Hello $name");
})
```

This will display "Hello " for the URL `/hello` and "Hello Rob" for the URL `/hello/Rob`.

If we wanted a default of "World," we can set an argument on the Route object that is returned from `get()` (and all the other routing methods):

```
$app->get("/hello[/{name}]", function ($request, $response, $args) {
$name = $request->getAttribute('name');
return $response->write("Hello $name");
})->setArgument('name', 'World');
```

This works exactly as you would expect.

The route arguments don't have to be a placeholder and you can set multiple route arguments. For example:

```
$app->get("/hello[/{name}]", function ($request, $response, $args) {
$name = $request->getAttribute('name');
$foo = $request->getAttribute('foo');
return $response->write("Hello $name, foo = $foo");
})->setArguments([
'name' => 'World,
'foo' => 'bar',
]);
```

Now, we have a foo attribute in our request, which is a per-route configuration option that you can do with as you wish - e.g. setting acl rules like this:

```
$app->get("/users", App\HelloAction::class)->setArgument("role", "userAdministrator");
```

Like This Article? Read More From DZone


Using Eloquent in Slim Framework


I Dare You Not to Fall In Love


How to Properly Setup RSpec


Free DZone Refcard
Low-Code Application Development

Topics: WEB DEV , SLIM FRAMEWORK , RUBY

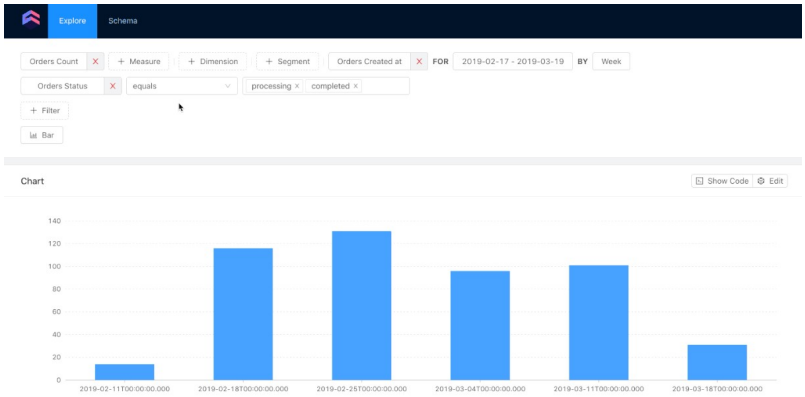
React Query Builder With Cube.js

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.



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/rea
ct";
const cubejsApi = cubejs("CUBEJS_TOKEN", { apiur
l: "CUBEJS_BACKEND_URL" });

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

Visualization Development

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