gatsby-source-mongodb

Source plugin for pulling data into Gatsby from MongoDB collections.

How to use

```
// In your gatsby-config.js
module.exports = {
  plugins: [
    /*
     * Gatsby's data processing layer begins with "source" plugins. Here we
     * setup the site to pull data from the "documents" collection in a local
     * MongoDB instance
     */
    {
      resolve: `gatsby-source-mongodb`,
      options: { dbName: `local`, collection: `documents` },
      query: { documents: { as_of: { $gte: 1604397088013 } } },
    },
  ],
}
```

multiple collections

```
// In your gatsby-config.js
module.exports = {
  plugins: [
    {
      resolve: `gatsby-source-mongodb`,
      options: { dbName: `local`, collection: [`documents`, `vehicles`] },
    },
  ],
}
```

Plugin options

- **connectionString**: if you need to use a connection string compatible with later versions of MongoDB, or for connections to MongoDB Atlas, you can enter the entire string, minus the `dbName` and `extraParams`. In this case, the authentication information should already be embedded in the string ex. `mongodb+srv://<USERNAME>:<PASSWORD>@<SERVERNANE>-fsokc.mongodb.net`. Pass `dbName` and `extraParams` as options per below.
- **dbName**: indicates the database name that you want to use
- **typePrefix**: the prefix you want to add to your entity types (dbName by default)

- **collection**: the collection name within Mongodb, this can also be an array for multiple collections
- **query**: add a query when retriving a collection. This is a key value object where key's are collection names, and value is the query object. Defaults to {} (i.e. the whole collection)
- **server**: contains the server info, with sub properties address and port ex. server: { address: `ds143532.mlab.com`, port: 43532 }. Defaults to a server running locally on the default port.
- **auth**: the authentication data to login a Mongodb collection, with sub properties user and password. ex. auth: { user: `admin`, password: `12345` }
- **extraParams**: useful to set additional parameters for the connection, like authSource, ssl or replicaSet (needed for connecting to MongoDB Atlas db as a service), ex: extraParams: { replicaSet: `test-shard-0`, ssl: `true`, authSource: `admin` }. These are the types of options that can be appended as query parameters to the connection URI: https://docs.mongodb.com/manual/reference/connection-string/#connections-connection-options
- **clientOptions**: for setting options on the creation of a `MongoClient` instance. By default, we pass `useNewUrlParser: true` to handle the various connection URI's necessary for newer versions of MongoDB Atlas, as well as `useUnifiedTopology: true` to opt in to using the MongoDB driver's new topology engine, which improves server discovery and monitoring. You can override the default by passing either an empty object literal or filled with other valid connection options. All options specified in the `MongoClient` documentation are valid for usage.
- **preserveObjectIds**: for preserving nested ObjectIDs within documents, set this to the Boolean `true`, ex: preserveObjectIds: `true`.

Mapping mediatype feature

Gatsby supports transformer plugins that know how to transform one data type to another e.g. markdown to html. In the plugin options you can setup "mappings" for fields in your collections. You can tell Gatsby that a certain field is a given media type and with the correct transformer plugins installed, your data will be transformed automatically.

Let's say we have a markdown field named `body` in our mongoDB collection `documents`. We want to author our content in markdown but want to transform the markdown to HTML for including in our React components.

To do this, we modify the plugin configuration in `gatsby-config.js` like follows:

```
module.exports = {
  plugins: [
    {
      resolve: `gatsby-source-mongodb`,
      options: {
```

```
        dbName: `local`,
        collection: `documents`,
        // highlight-start
        map: {
          documents: { body: `text/markdown` },
        },
        // highlight-end
      },
    },
  ],
}
```

The GraphQL query to get the transformed markdown would look something like this.

```
query ($id: String!) {
  mongodbCloudDocuments(id: { eq: $id }) {
    id
    name
    url
    body {
      childMarkdownRemark {
        id
        html
      }
    }
  }
}
```

How to query your MongoDB data using GraphQL

Below is a sample query for fetching all MongoDB document nodes from a db named **'Cloud'** and a collection named **'documents'**.

```
query {
  allMongodbCloudDocuments {
    edges {
      node {
        id
        url
        name
      }
    }
  }
}
```