Carga masiva en NodeJS usando un API como fuente de datos

Dario Alvarez Arteaga

Equipo JDC

Ingeniería Cloud: Arquitecturas para software como servicio

9 de marzo de 2020

El problema y la solución

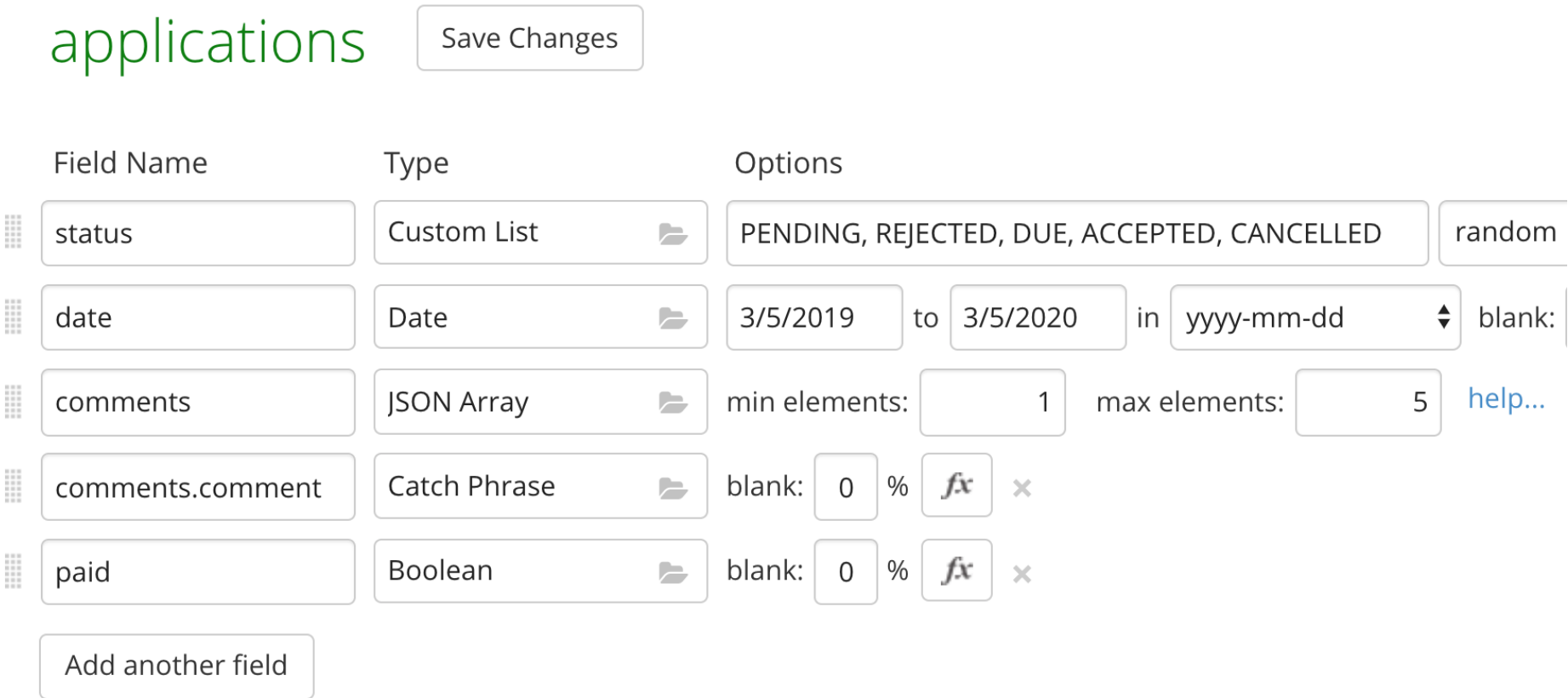
El problema

No existe (o no he encontrado) una herramienta que permita hacer la carga masiva en *NodeJS* teniendo en cuenta las relaciones entre modelos.

La solución

- *Mockaroo* como API generadora de datos *mock*.

- Paquetes *request* y *JSONStream* para leer los datos y convertirlo a un *stream* de datos en formato *JSON*.
- Paquete *stream-to-mongo-db* de Node para insertar el *stream* en formato *JSON* hacia la base de datos en *MongoDB*.


applications
Save Changes
Field Name
Type
Options
status
Custom List
PENDING, REJECTED, DUE, ACCEPTED, CANCELLED
random
date
Date
3/5/2019
to
3/5/2020
in
yyyy-mm-dd
blank:
comments
JSON Array
min elements:
1
max elements:
5
help...
comments.comment
Catch Phrase
blank:
0
%
fx
paid
Boolean
blank:
0
%
fx
Add another field

Editing API

Route

GET ▾ `/applications.json`

Handler Script

```
schema "applications"
generate 4000
```

Try in browser:

https://my.api.mockaroo.com/applications.json?key=c4e6e8c0

Try with cURL:

```
curl -H "X-API-Key: c4e6e8c0" https://my.api.mockaroo.com/applications.json
```

El problema de la solución

La solución aún tiene algunos problemas

- Se insertan los datos directamente en la base de datos en *MongoDB*, en lugar de usar *Mongoose*. No se usa la lógica implementada en el modelo y/o controlador de la aplicación.
- Decisión de insertar los datos *mock* solo cuando la base de datos esté vacía (o no?).
- Los **Trip** tienen un *ticker* con el formato YYMMDD-ABCD. No se pueden repetir los *ticker*.
- Es necesario crear las relaciones entre los elementos:
  - **Actor** no depende de nadie
  - **Trip** necesita un **Actor** de tipo *Manager*
  - **Application** necesita un **Actor** de tipo *Explorer* y un **Trip**.
  - La inserción debe ser síncrona: **Actor**, **Trip**, **Application**.

Dentro del fichero *index.js*

Usando el módulo *massiveLoad*

```
// Massive load section
console.log("== Starting massive loading ==");
massiveLoad.loadActorsFromApi(mongoose, mongoDBURI, false,
    'https://my.api.mockaroo.com/actors.json?key=c4e6e8c0', function () {

    massiveLoad.loadTripsFromApi(mongoose, mongoDBURI, false,
        'https://my.api.mockaroo.com/trips.json?key=c4e6e8c0', function () {

        massiveLoad.loadApplicationsFromApi(mongoose, mongoDBURI, false,
            'https://my.api.mockaroo.com/applications.json?key=c4e6e8c0', function () {

            console.log("== Finished massive loading ==");

        });

    });

});
```

Dentro del módulo *massiveLoad.js*

Insertar los datos *mock* solo cuando la base de datos esté vacía (o no?)

```
module.exports.loadActorsFromApi =
    function (mongoose, mongoDBURI, onlyIfEmptyCollection, apiUrl, next) {
    console.log("Starting loading Actors");
    var Actor = mongoose.model('Actors');

    Actor.estimatedDocumentCount({}, function (err, documentCount) {
        if (err) {
            res.send(err);
        } else if (onlyIfEmptyCollection && documentCount > 0) {
            console.log('DB already has ' + documentCount + ' actors.');
            next();
        } else {
            console.log('Empty DB Actors, loading initial data...');
```

Dentro del módulo *massiveLoad.js*

Haciendo uso de los paquetes *stream-to-mongo-db*, *request* y *JSONStream*

```
console.log('Empty DB Actors, loading initial data...');
// where the data will end up
const outputDBConfig = {
    dbURL: mongoDBURI,
    collection: 'actors'
};
// create the writable stream
const writableStream = streamToMongoDB(outputDBConfig);

// create readable stream and consume it
request(apiUrl)
    .pipe(JSONStream.parse('*'))
    .pipe(writableStream).on('finish', () => {
        console.log("Finished loading Actors");
        next();
    });
```

Dentro del módulo *massiveLoad.js*

Evitando que se repitan los *ticker*

```
Trip.find({}, ['ticker'], function (err, tickers) {
    tickers = tickers.map(item => item.ticker);
    // create readable stream and consume it
    request(apiUrl)
        .pipe(JSONStream.parse('*'))
        .on('data', function (data) {
            const alphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
            let now = moment().format("YYMMDD");
            var tickerCandidate = "",
                tickerValid = false;
            while (!tickerValid) {
                let randomletters = generate(alphabet, 4);
                tickerCandidate = `${now}-${randomletters}`;
                if (tickers.includes(tickerCandidate)) {
                    console.log("Ticker already exists: " + tickerCandidate);
                } else {
                    tickerValid = true;
                    tickers.push(tickerCandidate);
                }
            }
            data.ticker = tickerCandidate;
            data.manager = managers[Math.floor(Math.random() * managers.length)]._id;
            data.requirements = data.requirements.map(item => item.requirement);
        })
        .pipe(writableStream).on('finish', () => {
            console.log("Finished loading Trips");
            next();
        });
});
```

Dentro del módulo *massiveLoad.js*

Creando las relaciones de **Application** con **Actor** y **Trip**

```
Actor.find({role: "EXPLORER"}, ['_id'], function (err, explorers) {
    Trip.find({}, ['_id'], function (err, trips) {
        // create readable stream and consume it
        request(apiUrl)
            .pipe(JSONStream.parse('*'))
            .on('data', function (data) {
                // console.log(data);
                var explorer = explorers[Math.floor(Math.random() * explorers.length)]._id;
                var trip = trips[Math.floor(Math.random() * trips.length)]._id;
                data.explorer = explorer;
                data.trip = trip;
                data.comments = data.comments.map(item => item.comment);
            })
            .pipe(writableStream).on('finish', () => {
                console.log("Finished loading Applications");
                next();
            });
    });
});
```

Gracias

Carga masiva en NodeJS usando un API como fuente de datos

Dario Alvarez Arteaga

Equipo JDC

Ingeniería Cloud: Arquitecturas para software como servicio

9 de marzo de 2020