rest api for nedb database, based on express http server

58 commits 2 branches 0 releases 3 contributors MIT

Branch: master New pull request Find file Clone or download

bi-tm 1.2.4 Latest commit d29238f on Apr 15

test	improvement of $filter parser	4 months ago
.gitignore	initialization	11 months ago
.npmignore	initialization	11 months ago
LICENSE	initialization	11 months ago
README.md	More typos ans small enhancements	4 months ago
filter.js	improvement of $filter parser	4 months ago
index.js	improvement of $filter parser	4 months ago
order.js	initialization	11 months ago
package.json	1.2.4	4 months ago

README.md

express-nedb-rest

REST API for NeDB database, based on express HTTP server.

Recently i found the NeDB-project of Louis Chatriot. He developed a simple and very fast in-memory database (thank you!). I like it's zero administration and easy integration into nodejs application. There is no need to start a daemon process and to communicate with it. Unfortunately i found no RESTful web API for this database, so i implement own by my own.

My module is built on ExpressJS server framework and provides an express Router object. This can be integrated easily into any express application as middleware.

The API enables client sided javascript components to access database content via HTTP RESTful calls. This can be used i.e. for HTML5 applications.

Installation

You can download the source from Github or install it with npm:

```
npm install express-nedb-rest
```

Quick start

Following code snippet starts an express server, which serves nedb api at port 8080.

```
var express = require('express');
var nedb = require('nedb');
var expressNedbRest = require('express-nedb-rest');

// setup express app
var oApp = express();

// create  NEDB datastore
var datastore = new nedb({ filename: "test.db",  autoload: true });

// create rest api router and connect it to datastore
var restApi = expressNedbRest();
restApi.addDatastore('test', datastore);

// setup express server to serve rest service
oApp.use('/', restApi);

oApp.listen(8080, function () {
    console.log('you may use nedb rest api at port 8080');
});
```

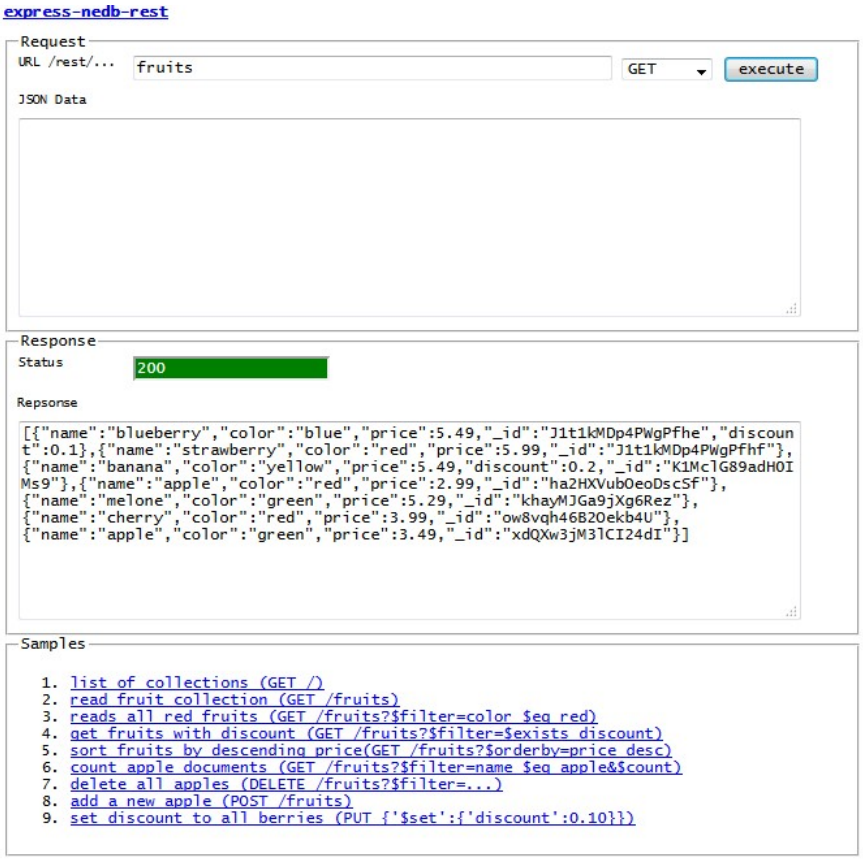
After starting the sample server, you can request a list of nedb datastores at `http://localhost:8080/` . You will get a response like:

```
[
    {"name":"test","link":"http://localhost:8080/test"}
]
```

For further testing you should use a REST client (i.e. postman or use my primitive test tool in path test/test.js).

Test tool

In filepath `test` you will find a test tool `test.js` . You can start it with command `node test/test.js` . It creates an express HTTP server and provides an `index.html` as web frontend. This frontend contains a form, in which you may set HTTP method, url and body text. You may execute the different HTTP methods (GET, POST, PUT, DELETE) and you will see the response content.



JavaScript module

The command `require('express-nedb-rest')` supplies a constructor function. It creates an express Router object. The router can be used as express middleware.

Methods

- constructor(options?)
  The constructor accepts an object as optional parameter.
  Currently there is only one option:
  - convertToDate:boolean
    true (default)= if as JSON string contains a date (ISO-8601), the string will be converted JavaScript Date object Please see section Date Objects.
- addDatastore(collection:string, store:nedb)
  Register a NeDB database for rest api. The NeDB database can be accessed under the collection name
- setValidator(callback:function)
  Register a callback function which will be called as validator before each NeDB database call.
  The validator function should have the typical expressJS signature (req, res, next)

API schema

The module can be connected to multiple NeDB data storages, which are called *collections*. Each CRUD command is a combination of a HTTP method (GET, POST, PUT, DELETE), URL and HTTP-body. The following table gives a quick overview of possible commands.

URL	Method	Notes
/	GET	get list of collections (= datastores)
/:collection	GET	search documents in a collection (uses query parameter $filter $orderby)

URL	Method	Notes
/:collection/:id	GET	retrieve a single document
/:collection	POST	create a single document
/:collection/:id	PUT	update a single document
/:collection	PUT	update multiple documents (uses query parameter $filter)
/:collection/:id	DELETE	remove single a document
/:collection	DELETE	remove multiple documents (uses query parameter $filter)

Creating Documents

To create a document, use a POST call and put the document into HTTP body. You can only insert one document per call. Each document must have a unique key value, which is named '_id'. If you don't define an _id, NeDB will generate a 16 character long string as _id. Please refer to NeDB documentation. On success the server will respond with status code 201, and the body contains the created document as JSON string.

Reading Documents

Read operation are done by HTTP GET calls. You can read a single document by appending the document _id to the URL. In this case the server will respond with the document as JSON string.

```
HTTP GET /fruits/J1t1kMDp4PWgPfhe
```

You can also query multiple documents and set a $filter as parameter. In that case the response contains an array of document objects (JSON formatted). You may also get an empty array, if no document matches the filter. The result can be sorted with parameter $orderby

```
HTTP GET /fruits?$filter=$price $lt 3.00&$orderby=price
```

Updating Documents

Updating operations are done by HTTP PUT calls. You can update a single document by appending the document key (_id) to URL. You must provide the document in HTTP body as JSON string. You cannot change key field (_id). The document will be completely overwritten with the new content.

If you don't want to update every field of the document, but only change some of them, you have to use a special NeDB syntax. There are operations $set, $unset, $inc and more, to update a field.

```
HTTP PUT /fruits/J1t1kMDp4PWgPfhe
{ $set: { discount: 0.10 } }
```

You can also update multiple documents by calling a PUT command without _id. You should define a $filter, otherwise all documents are changed. Changing multiple documents makes only sense in combination with update operations like $set. Otherwise all documents of a collection will have the same content.

```
HTTP PUT /fruits?$filter=name $regex berry
{ $set: { discount: 0.10 } }
```

Deleting Documents

Documents can be deleted by HTTP DELETE calls. You can delete a single document by appending the document key (_id) to the URL.

```
HTTP DELETE /fruits/J1t1kMDp4PWgPfhe
```

If you omit the _id, you must define $filter parameter, to specify a subset of documents. Otherwise the server will respond with error status 405. This shall protect you to delete all documents by accident.

```
HTTP DELETE /fruits?$filter=name $regex berry
```

Query parameter $filter

The $filter parameter is used, to define a subset of documents of a collection. Filter may be used for reading (GET), updating (PUT) and deleting (DELETE) commands.

A filter consists of one or more conditions, which are linked with logical and/or operations. Filters are set by the $filter parameter. The string will be parsed and transformed to a NeDB filter object. Filters has format . Values may be a String, Boolean, Number, Date or Array.

If you compare with a date value, please define it as ISO-8601 string (i.e. 2017-04-06T08:39:44.016Z). Please refer to section "Date Objects"

For the operators $in and $nin an array must be given as value. Currently this array cannot obtain a single value. Arrays are delimited by `,`. Another constraint is that an array can only contain a single type of either String of Number. The array `1,2,hello` will not work.

Here is a list of valid operations. For more informations please consult NeDB documentation.

operators	description	example
$eq $ne	equal, not equal	/fruits?$filter=color $eq red
$lt $lte	less than, less than or equal	/fruits?$filter=price $lt 2.00
$gt $gte	greater than, greater than or equal	/fruits?$filter=price $gte 5.00
$exists	checks whether the document posses the property field.	/fruits?$filter=$exists discount
$regex	checks whether a string is matched by the regular expression.	/fruits?filter=name $regex foo
$and $or	logical and/or oparator	/fruits?$filter=name $eq apple $and color $eq red
$in $nin	member of, not member of	/fruits?$filter=name $in apple,banana
$not	not operator	/fruits?$filter=$not name $regex foo

Query parameter $orderby

You may sort the result of a query with "$orderby" parameter. You can use it in reading (GET) operations only. The parameter may contain multiple fieldnames concatenated by commas (,). Each fieldname can be followed by keyword `asc` or `desc` to define sorting direction. Ascending is default direction, so you may omit it.

Examples:

```
HTTP GET /fruits?$orderby=price
```

```
HTTP GET /fruits?$filter=color $eq red&$orderby=price
```

Query parameter $count

If you append $count parameter to a query, the server returns the number of of matching documents instead of a result set. You can use this parameter in reading (GET) operations only. The server responds with a number (no JSON object or array).

Example: `HTTP GET /fruits?$filter=name $eq apple&$count`

Query parameter $skip and $limit

If you want to fetch results in several packages, you may use pagination parameters $skip and $limit. They should be used together with $orderby parameter.
Parameter $skip sets the count of documents, which will be deleteted from the beginning of result set.
Parameter $limit sets maximal count of documents in the result set.
You can use this parameter in reading (GET) operations only.

Example: `HTTP GET /fruits?$filter=name $eq apple&$skip=1&$limit=2`

Query parameter $single

If you read from collections with HTTP GET, the result will be always an array of documents, even if you use query parameter $limit=1, or only one docment matches the $filter.

If you prefer to get a single object but not an array, you must use query parameter $single instead. The NeDB database will be queried with function ´findOne´, and you will get only one document as JSON object. If your query finds no document, you will get a 404-error code, instead of an empty array.

Example: `HTTP GET /fruits?$filter=name $eq apple&$single`

Date Objects

There is no general specification how to define a date in JSON string. Nevertheless you want to set date-time values in documents. As solution i use a modified JSON-parser. If you set an ISO-8601 string into document's JSON (i.e. { "date":"2017-04-06T08:39:44.016Z" }), the string is parsed to JS Date object. In NeDB the date field will be a Date object instead of String.

I added this special feature in version 1.2.0. In older releases all strings were transfered to NeDB without changes. If you want to switch back to the old behaviour, you have to set an option when instantiating express-nedb-rest object: `var restApi = expressNedbRest({convertToDate:false});`