

Standardized API Development

Nicolas De Smyter

FDS

March 17, 2017

- 1 Intro
- 2 Installation and configuration
 - First setup
 - Configure database
 - Configure model
 - User implemented method
 - Start API server
- 3 Query data
- 4 Conclusion
 - Further reading

Loopback

LoopBack is a highly-extensible, open-source Node.js framework that enables you to create dynamic end-to-end REST APIs with little or no coding.

dive

[Show/Hide](#) | [List Operations](#) | [Expand Operations](#)

PATCH	/dives	Patch an existing model instance or insert a new one into the data source.
GET	/dives	Find all instances of the model matched by filter from the data source.
PUT	/dives	Replace an existing model instance or insert a new one into the data source.
POST	/dives	Create a new instance of the model and persist it into the data source.
PATCH	/dives/{id}	Patch attributes for a model instance and persist it into the data source.
GET	/dives/{id}	Find a model instance by {{id}} from the data source.
HEAD	/dives/{id}	Check whether a model instance exists in the data source.
PUT	/dives/{id}	Replace attributes for a model instance and persist it into the data source.
DELETE	/dives/{id}	Delete a model instance by {{id}} from the data source.
GET	/dives/{id}/exists	Check whether a model instance exists in the data source.

First setup

```
$ npm install -g loopback-cli  
$ lb
```

Configure database

```
$ lb datasource
? Enter the data-source name: postgres-db
? Select the connector for oracledb: PostgreSQL
Connector specific configuration:
? Connection String url to override other settings:
postgres://postgres:postgres@localhost/mantis
? host: localhost
? port: 5432
? user: postgres
? password: *****
? database: mantis
? install loopback-connector-postgresql@^2.4 Yes
```

Configure model

```
$ lb model
```

```
? Enter the model name: dive
```

```
? Select the data-source to attach dive to:
```

```
postgres-db
```

```
? Select model's base class PersistedModel
```

```
? Expose dive via the REST API? Yes
```

```
? Custom plural form (used to build REST URL):
```

```
dives
```

```
? Common model or server only? common
```

```
Let's add some dive properties now.
```

Configure model properties

Enter an empty property name when done.

? Property name: **id**

invoke loopback:property

? Property type: **number**

? Required? **Yes**

? Default value [leave empty for none]:

Link models

```
$ lb relation
```

```
? Name of the model to create the relationship  
from: dive
```

```
? Relation type: has many
```

```
? Name of the model to create a relationship with:  
participant
```

```
? Name for the relation: participants
```

```
? Custom foreign key: dive_id
```

```
? Whether a "through" model is required? No
```


Existing methods

Every model:

- create
- find
- findOne
- findById
- updateAll
- createUpdates
- destroyAll
- ...

Remote method

```
Dive.divesByCountry = function (country, cb) {
  var Location = app.models.location;

  Location.find({fields: ["id"], where: {location: country}}, function (err,
    locations) {

    var locationIds = [];
    for (var i = 0, length = locations.length; i < length; i++) {
      locationIds.push(locations[i].id);
    }
    Dive.find({where: {location_id: {"inq": locationIds}}}, cb);
  })
};

Dive.remoteMethod('divesByCountry', {
  accepts: {arg: 'country', type: 'string'},
  returns: {arg: 'dives', type: 'Dive[]'},
  description: 'List the dives that are organized in a specific country',
  http: {verb: 'get'}
});
```

Access control

- `$ lb acl`
- Built-in, using authentication tokens
- Security per method or end-point
- `*`, `READ`, `WRITE`, `EXECUTE`
- `DENY`, `ALLOW`
- `$everyone`, `$authenticated`, `$unauthenticated`, `$owner`

Start API server

```
$ node .
```

In debugging mode:

```
$ DEBUG=loopback:connector:postgresql node .
```

Query data

Use the built-in explorer: <http://localhost:3000/explorer>

dive

[Show/Hide](#)[List Operations](#)[Expand Operations](#)

PATCH	/dives	Patch an existing model instance or insert a new one into the data source.
GET	/dives	Find all instances of the model matched by filter from the data source.
PUT	/dives	Replace an existing model instance or insert a new one into the data source.
POST	/dives	Create a new instance of the model and persist it into the data source.
PATCH	/dives/{id}	Patch attributes for a model instance and persist it into the data source.
GET	/dives/{id}	Find a model instance by {{id}} from the data source.
HEAD	/dives/{id}	Check whether a model instance exists in the data source.
PUT	/dives/{id}	Replace attributes for a model instance and persist it into the data source.
DELETE	/dives/{id}	Delete a model instance by {{id}} from the data source.
GET	/dives/{id}/exists	Check whether a model instance exists in the data source.

Filter

- fields
- where
- include
- order
- offset
- limit

Query data - Request

GET /dives Find all instances of the model matched by filter from the data source.

Response Class (Status 200)
Request was successful

Model | **Example Value**

```
[
  {
    "id": 0,
    "name": "string",
    "date": "2017-03-10T10:23:02.134Z",
    "location_id": 0
  }
]
```

Response Content Type application/json ▼

Parameters

Parameter	Value	Description	Parameter Type	Data Type
filter	<input type="text"/>	Filter defining fields, where, include, order, offset, and limit	query	string

Try it out!

Query data - Response

Curl

```
curl -X GET --header 'Accept: application/json' 'http://localhost:3000/api/dives'
```

Request URL

```
http://localhost:3000/api/dives
```

Response Body

```
[
  {
    "id": 1,
    "name": "Kerstduik",
    "date": "2016-12-25T11:00:00.000Z",
    "location_id": 1
  },
  {
    "id": 2,
    "name": "Oudejaarsduik",
    "date": "2016-12-31T22:00:00.000Z",
    "location_id": 2
  },
  {
    "id": 3,
    "name": "Nieuwjaarsduik",
    "date": "2017-01-01T09:00:00.000Z",
    "location_id": 3
  },
  {

```

Response Code

```
200
```


Further reading

- Boot scripts
- Input validation
- Changes to core code needed for geographic support

- Betalend

www-03.ibm.com/software/products/en/api-connect

Conclusion

- + Easy to configure and use
- + Different database types supported
- + Highly configurable
- + Good and useful documentation with examples
- + Input validation
- No native geographic support (code changes)

The End

Questions ?

All code and presentation:

<https://github.com/ndsmyter/loopback-brownbag>

Nicolas.DeSmyter@esfds.com