Nick Duffy
Fatherhood, data, Python, NY Mets
Mar 23, 2015 · 4 min read

Getting started with LoopBack and SQL Server

In this tutorial, we'll talk about getting started with LoopBack and SQL Server. We'll create an API using an existing SQL Server table as the backend. Let's jump right in.

Prerequisites

- You have Node.js installed
- You have some version of SQL Server installed

SQL Server

You know SQL Server.

> SQL Server is Microsofts RDMS offering for OLTP, data warehousing, business intelligence and analytics.

The data

We're going to be using an existing SQL Server table in our database. This should be a very common scenario when working with LoopBack and SQL Server as you are likely not creating your tables in SQL Server specifically while creating your LoopBack application.

We'll use the excellent generatedata.com site to seed our table with 100 rows.

```
CREATE TABLE [contacts] (
    [id] INTEGER NOT NULL IDENTITY(1, 1),
    [first_name] VARCHAR(255) NULL,
    [last_name] VARCHAR(255) NULL,
    PRIMARY KEY ([id])
);
GO
```

```
INSERT INTO contacts([first_name],[last_name]) VALUES('Malachi','Burch'),('Stone','Horne'),
INSERT INTO contacts([first_name],[last_name]) VALUES('Lyle','Martin'),('Isaac','Huff'),('S
INSERT INTO contacts([first_name],[last_name]) VALUES('Theodore','Mcdonald'),('Luke','Hicks
INSERT INTO contacts([first_name],[last_name]) VALUES('Mason','Paul'),('Xander','Nunez'),('
INSERT INTO contacts([first_name],[last_name]) VALUES('Fuller','Vaughan'),('Davis','White')
INSERT INTO contacts([first_name],[last_name]) VALUES('Cedric','House'),('Honorato','Rosari
```

getting_started_with_loopback_and_sql_server-create_table.sql hosted with ❤ by **GitHub** **view raw**

Alright, now that we have our already existing table example setup, let's move on to LoopBack.

LoopBack

> LoopBack is a highly-extensible, open-source Node.js framework

LoopBack allows you to do many things like:

- Quickly create dynamic end-to-end REST APIs.
- Connect devices and browsers to data and services.
- Use Android, iOS, and AngularJS SDKs to easily create client apps.
- Add-on components for push, file management, 3rd-party login, and geolocation.
- Use StrongLoop Arc to visually edit, deploy, and monitor LoopBack apps.
- LoopBack API gateway acts an intermediary between API consumers (clients) and API providers to externalize, secure, and manage APIs.
- Runs on-premises or in the cloud

We are going to use LoopBack to:

- Connect to our SQL Server database.
- Discover the schema of our already existing table and create a LoopBack model based upon the schema.
- Create a REST based API using this model.

```
npm install -g strongloop
```

getting_started_with_loopback_and_sql_server-install_strongloop.sh hosted with ❤ by GitHub view raw

Well that was easy! Now let's scaffold our new LoopBack API application.

Run the LoopBack application generator and answer a few quick questions.

```
$ slc loopback
     _-----_
    |       |    .--------------------------.
    |--(o)--|    |  Let's create a LoopBack |
   `---------´   |       application!       |
    ( _´U`_ )    '--------------------------'
    /___A___\
     |  ~  |
   __'.___.'__
 ´   `  |° ´ Y `

? What's the name of your application? loopback-sqlserver-example
? Enter name of the directory to contain the project: loopback-sqlserver-example
```

getting_started_with_loopback_and_sql_server-create_loopback_app.sh hosted with ❤ by **GitHub** **view raw**

After the generator runs and npm installs all the necessary packages, you'll receive a message like this.

```
Next steps:

  Change directory to your app
    $ cd loopback-sqlserver-example

  Create a model in your app
    $ slc loopback:model

  Optional: Enable StrongOps monitoring
    $ slc strongops

  Run the app
    $ slc run .
```

getting_started_with_loopback_and_sql_server-next_steps.sh hosted with ❤ by **GitHub** **view raw**

loopback-connector-mssql.

```
npm install loopback-connector-mssql --save
```

getting_started_with_loopback_and_sql_server-install_mssql.sh hosted with ❤ by GitHub view raw

Connect to SQL Server

Now we need to point our LoopBack application to our SQL Server. In your loopback-sqlserver-example directory, open up the server/datasources.js file. Initially there is only a listing for the in-memory database, but we want to add a reference to our SQL Server.

Edit the file to look like this, of course, replacing with your SQL Server host, username and password.

```
{
  "db": {
    "name": "db",
    "connector": "memory"
  },
  "mySqlServer": {
    "host": "192.168.0.23",
    "port": 1433,
    "database": "myDb",
    "username": "username",
    "password": "password",
    "name": "mySqlServer",
    "connector": "mssql",
    "schema": "dbo"
  }
}
```

getting_started_with_loopback_and_sql_server-setup_mssql.json hosted with ❤ by GitHub view raw

Now we're ready to build our LoopBack model for our SQL Server table.

LoopBack table model

LoopBack uses JSON models to define tables in the application. While we could create the JSON by hand, we want to use our already existing table to automatically generate it for us. Thankfully LoopBack offers the discoverSchema method which will auto generate the JSON for us based upon our already existent SQL Server table.

```
var app = require('./server');
var dataSource = app.dataSources.mySqlServer;

dataSource.discoverSchema('contacts', {
  owner: 'dbo'
}, function(err, schema) {
  console.log(JSON.stringify(schema, null, '  '));
});
```

getting_started_with_loopback_and_sql_server-discover_table.js hosted with ❤ by **GitHub** **view raw**

First we get a reference to our app and then to the mySqlServer data source we created in the datasource.js file. Then we tell LoopBack to go get the schema of our table and output the JSON model to the console.

Run this script and you'll get LoopBack's JSON model representation of your table.

```
$ node server/discover.js
{
  "name": "Contacts",
  "options": {
    "idInjection": false,
    "mssql": {
      "schema": "dbo",
      "table": "contacts"
    }
  },
  "properties": {
    "id": {
      "type": "Number",
      "required": true,
      "length": null,
      "precision": 10,
      "scale": 0,
      "id": 1,
      "mssql": {
        "columnName": "id",
        "dataType": "int",
        "dataLength": null,
        "dataPrecision": 10,
        "dataScale": 0,
        "nullable": "NO"
      }
    },
```

```
        "scale": null,
        "mssql": {
          "columnName": "first_name",
          "dataType": "varchar",
          "dataLength": 255,
          "dataPrecision": null,
          "dataScale": null,
          "nullable": "YES"
        }
      },
      "lastName": {
        "type": "String",
        "required": false,
        "length": 255,
        "precision": null,
        "scale": null,
        "mssql": {
          "columnName": "last_name",
          "dataType": "varchar",
          "dataLength": 255,
          "dataPrecision": null,
          "dataScale": null,
          "nullable": "YES"
        }
      }
    }
}
```

getting_started_with_loopback_and_sql_server-discover_schema.json hosted with ❤ by **GitHub** **view raw**

Cool! LoopBack just autogenerated a model of our contacts table based upon the already existing table.

Let’s get that model integrated into our app. We need to create two files to represent our contacts table in LoopBack.

```
$ mkdir -p common/models
$ touch common/models/contacts.js
$ touch common/models/contacts.json
```

getting_started_with_loopback_and_sql_server-create_files.sh hosted with ❤ by **GitHub** **view raw**

The JSON file will hold our model definition and JavaScript file will export our model. There are more uses for the JavaScript file which we'll get into in future posts.

First, copy the JSON we generated above into the contacts.json file.

Next, drop this simple code into the contacts.js file.

```
1  module.exports = function(Contact) { };
```

getting_started_with_loopback_and_sql_server-export_contact.js hosted with ❤ by **GitHub** **view raw**

That's it for creating our LoopBack model. Now we can tell LoopBack to use our model in the API.

Create the API

We need to tell LoopBack to use our newly created Contact model in the API. To do so, we open up server/model-config.json and add an entry for our model.

```
1  {
2    "_meta": {
3      "sources": [
4        "loopback/common/models",
5        "loopback/server/models",
6        "../common/models",
7        "./models"
8      ]
9    },
10   "User": {
11     "dataSource": "db"
12   },
13   "AccessToken": {
14     "dataSource": "db",
15     "public": false
16   },
17   "ACL": {
18     "dataSource": "db",
19     "public": false
20   },
21   "RoleMapping": {
22     "dataSource": "db",
23     "public": false
24   }
```

```
      "public": false
    },
    "Contacts": {
      "dataSource": "mySqlServer",
      "public": true
    }
  }
```

getting_started_with_loopback_and_sql_server-data_sources.json hosted with ❤ by GitHub view raw

model-config.json already has some definitions in it for the LoopBack base models. We'll add our Contacts model at the end. Note that we set "public": true to tell LoopBack to expose our model via the API.

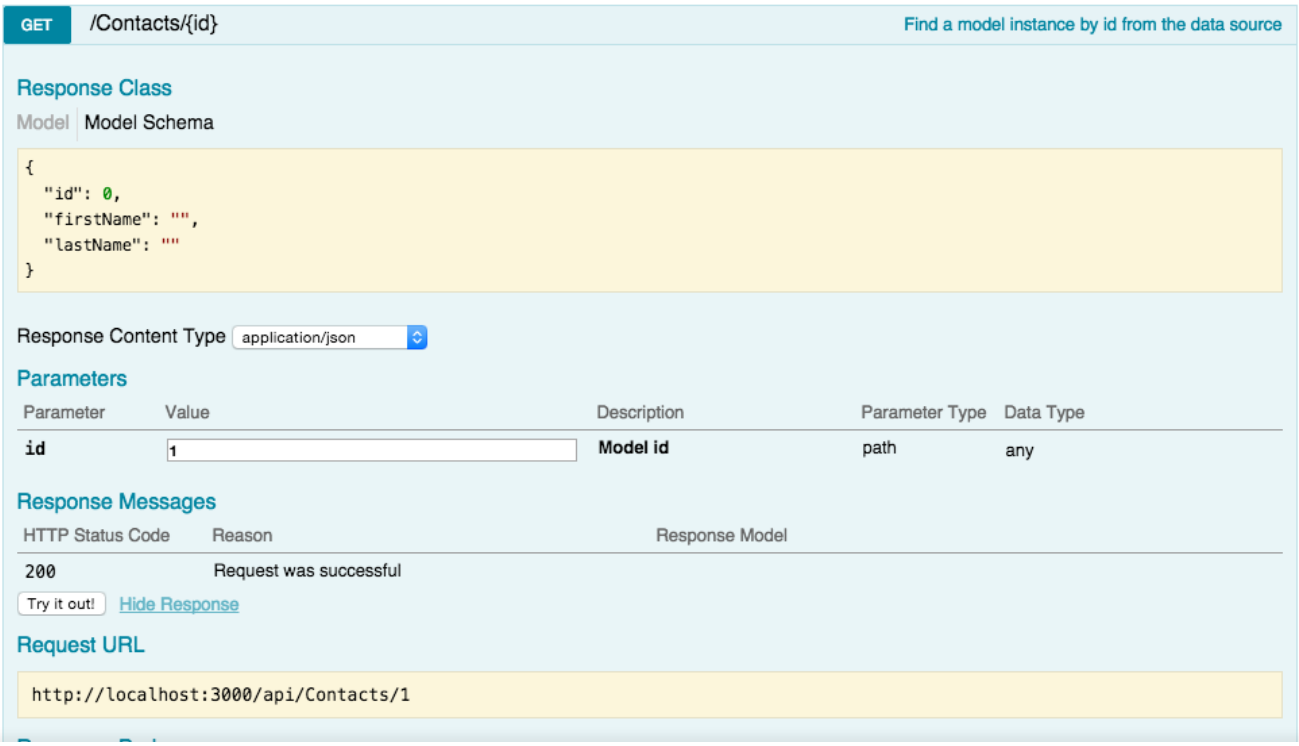
Run the API

Now it's time to fire up our API!

```
$ node server/server.js
```

getting_started_with_loopback_and_sql_server-run_server.sh hosted with ❤ by GitHub view raw

Browse to http://localhost:3000/explorer to see your API in action. Try to retrieve some data using the explorer and you can see that we have an working API!



Response Code

200

Conclusion

It doesn't take much to get an API up and running with LoopBack, even if you need to connect to and discover and already existing table in SQL Server—or in any database for that matter!

Nodejs | Sql Server | Loopback

More from Nick Duffy

Setup Jenkins CI with your Node.js LoopBack API and Bitbucket

Nick Duffy
6 min read

8

Also tagged Nodejs

Express.js and AWS Lambda — a serverless love story

Slobodan Stojanović
6 min read

2.2K

More from Nick Duffy

Supervisor reread and update

I needed to add some options to gunicorn running under supervisor the other day. I ran gunicorn from the command line with the new…

Nick Duffy
1 min read

2

Responses

Conversation with Nick Duffy.

Gregor Schmitz
Jul 20, 2017

don’t forget to enable TCP/IP on SQL Server Configuration Manager under the tab Network configuration. else sql server will not listen to port 1433

1 response

Nick Duffy
Nov 20, 2017

Thanks for the note.

Never miss a story from **Nick Duffy**

Conversation with Nick Duffy.

Christopher Bonn
Nov 9, 2016

Any chance you could update this to show moving the built in user model to the MSSQL db? Would make it far more useful. Even better would be if you extended the built-in user object and then showed it being persisted in the MSSQL db.

1 response

Nick Duffy
Nov 26, 2016

Thanks for the response. When I have the time and access to a Windows computer, I'll try to update the article.

Conversation with Nick Duffy.

Alex Broadchief
Nov 18, 2017

Great, thanks. But that article is a bit short minded. Yes, it is easy to connect and even use, but all the problematic details are hidden (as usual—seems people just connect to somewhere and stop using after that …). What about auto_increment ids? What about handling other MSSQL Based problems? Any chance to extend the article to be a bit more "competent" (thats just about the article, not you, so no offence).

1 response

Nick Duffy
Nov 20, 2017

Thanks for your response. The intent wasn't to provide an entire solution, but to get you started, hence the "Getting Started" title. At any rate, I no longer use SQL